

NETWORK PROGRAMMING

IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

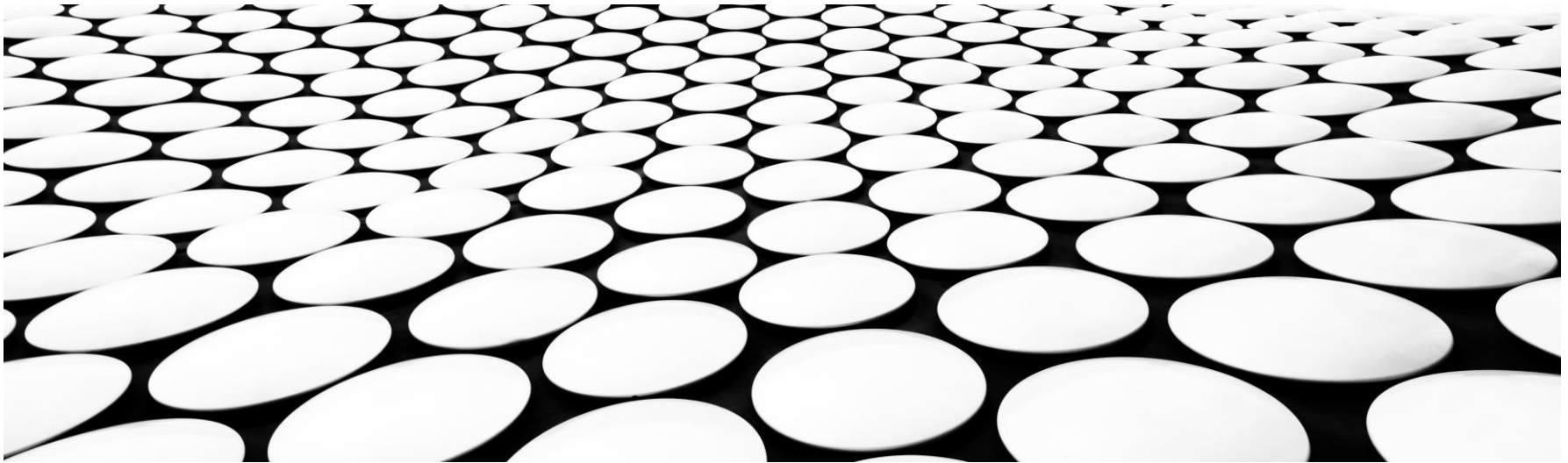
IT DEPT - OFFICE NO. 312

2024



كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

LECTURE 0

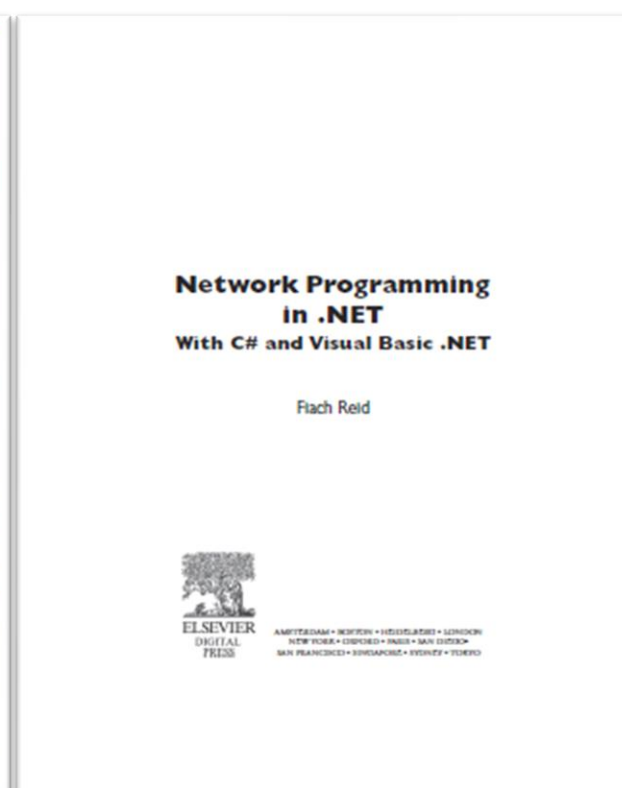
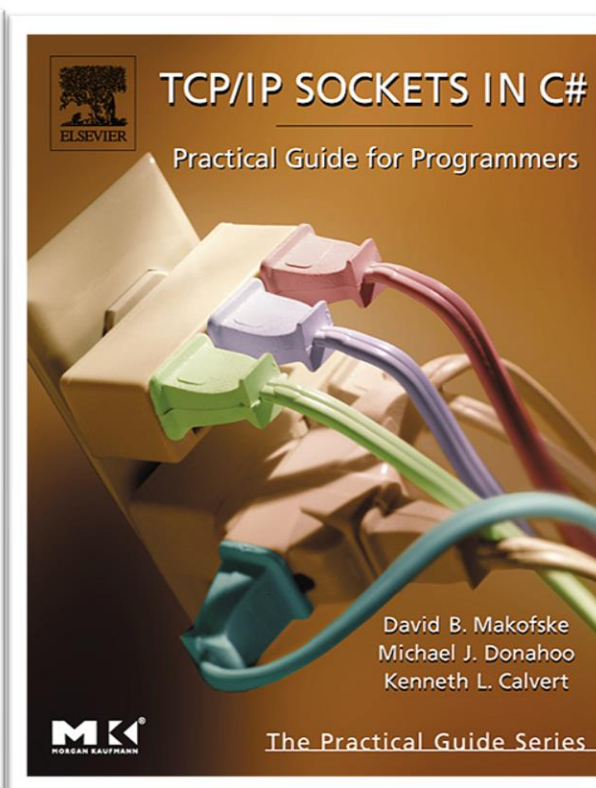
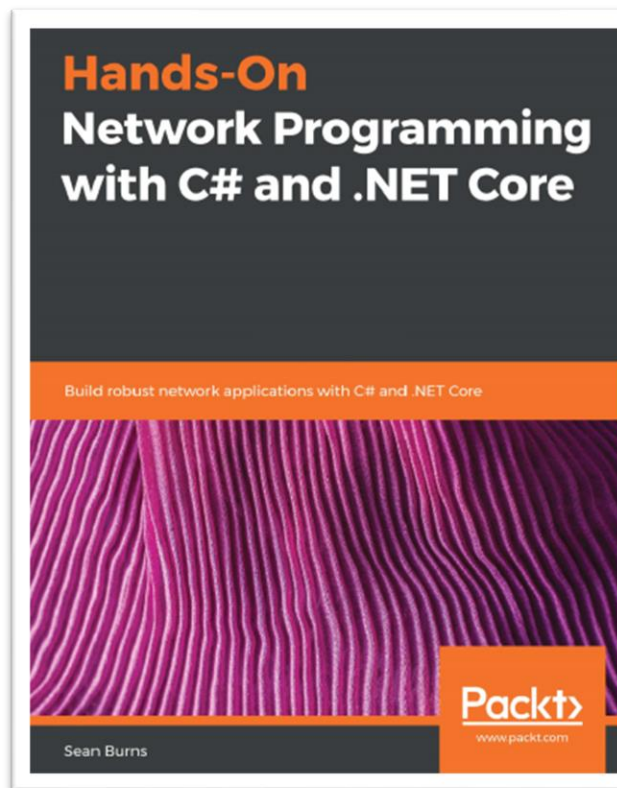




RESOURCES

- Book1: Hands-On Network Programming with C# and .NET Core, 2019.
- Book2: Network Programming in .NET With C# and Visual Basic .NET, Fiach Reid.
- Book3: TCP/IP Sockets in C# Practical Guide for Programmers
- Book:4: C# Network Programming, by Richard Blum
- Lecture notes.

BOOKS





YOUR BEST STRATEGY – ROADMAP TO SUCCESS IN THIS COURSE

- Come to every lecture
- Read the topics related to network protocols and network programming
- Do not wait till the last minute to prepare for your exam or to work on a project
- Enjoy !



COURSE CONTENT

- Introduction
- Streams
- Serialization
- Threading
- Socket Programming
 - Client/server
 - Web server
 - DNS
 - Peer to peer
 - Multicast and broadcast
- Project



LECTURE 0

- **What is Network Programming?**
- **Network Elements and protocols revision.**
- **C# Introduction**
- **C# Streams**

NETWORK PROGRAMMING

- Network Programming involves writing programs that communicate with other programs across a computer network.
- We use the Socket API .
- It can be achieved by any programming language, but during our course we will use C#.
- Example of programs:
 - Text, Audio and video chat.
 - Video broadcast and multicast.
 - Web browser
 - File exchange over network.
 - telnet (remote machine login)

WHAT'S IS A NETWORK?

- The term **network** can refer to any interconnected group or system.
- A **computer network** is composed of multiple computers connected together using a telecommunication system.
- “...communication system for connecting end-systems”
- End-systems or “hosts”
 - PCs
 - dedicated computers
 - network components
- Interconnection may be any medium capable of communicating information:
 - Copper wire
 - Lasers (optical fiber)
 - Radio /Satellite link
 - Cable (coax)
- Example: Ethernet.



WHY NETWORK?

- Sharing resources
 - Resources become available regardless of the user's physical location (server based, peer2peer)
- Load Sharing/utilization
 - Jobs processed on least crowded machine
- High reliability
 - Alternative sources for Info.

WIDE VARIETY TYPES OF NETWORKS

■ Circuit switched

- dedicated circuit per call
- performance (guaranteed)
- call setup required
- telephone system

■ Packet switched:

- data sent through the net in discrete “chunks”
- user A, B packets *share* network resources
- resources used *as needed*
- store and forward: packets move one hop at a time
- The Internet (TCP/IP)

WHAT IS INTERNET?

What is internet?

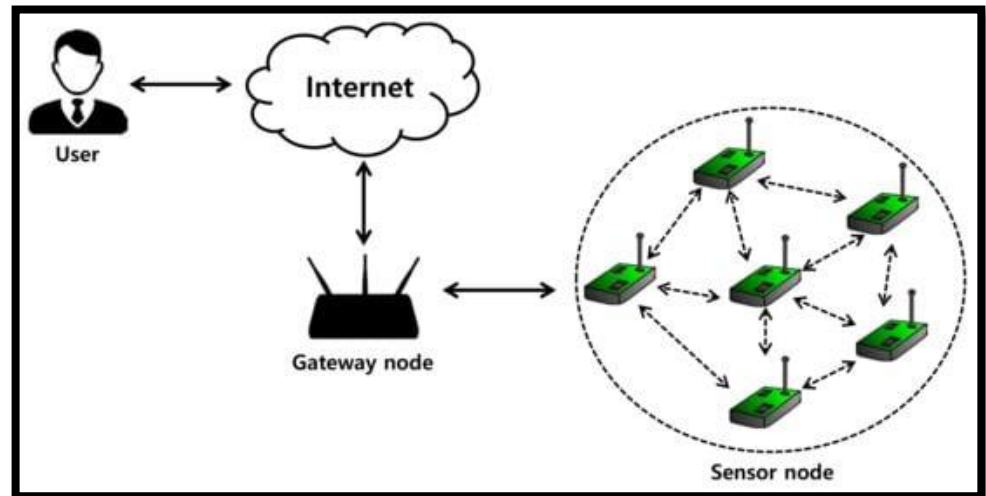
- Network of networks
- A global internet based on IP protocol

Internet applications:

- Email
- File transfer
- File sharing
- Resource distribution (DNS)
- World wide web
- Video conference
- Gaming

NEW EMERGING NETWORKS !

- Sensor Networks
 - Small devices equipped by sensor and connected to the internet
 - May include an environmental actuation
- Lots of them (density)
- Cheap unreliable elements
- Run on batteries
- Location becomes a key attribute
- Information sensing around users



LAN & WAN

- LAN : connects computers that are physically close together (< 1 mile (1.6 KM)).
 - high speed
 - multi-access
 - common technologies: Ethernet 10 Mbps, 100Mbps
- WAN connects computers that are physically far apart. “long-haul network”.
 - Slower than a LAN.
 - Less reliable than a LAN.
 - point-to-point
 - Technologies: telephone lines , Satellite communications

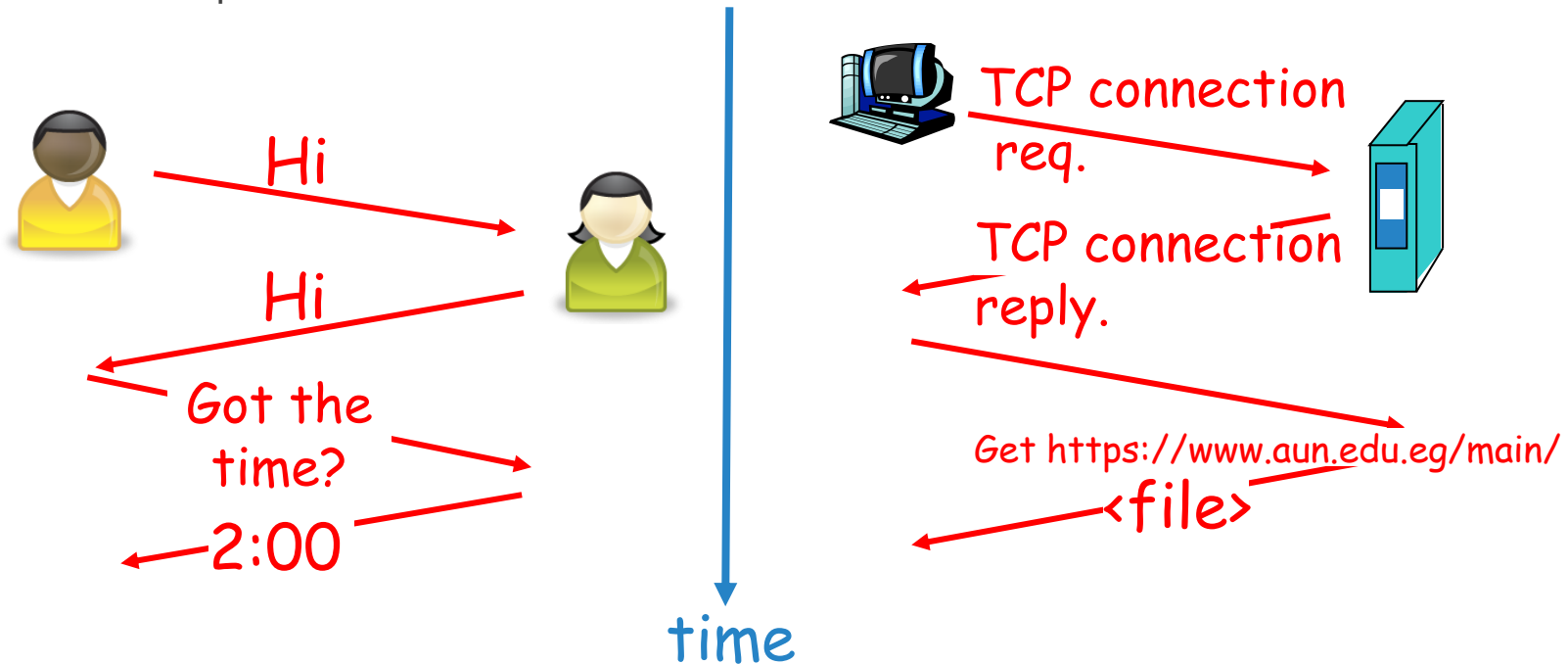


NETWORK PROTOCOLS



WHAT'S A PROTOCOL?

- a human protocol and a computer network protocol:



protocols define syntax , semantics and timing information required for entities to communicate (understand each other)

LAYERING

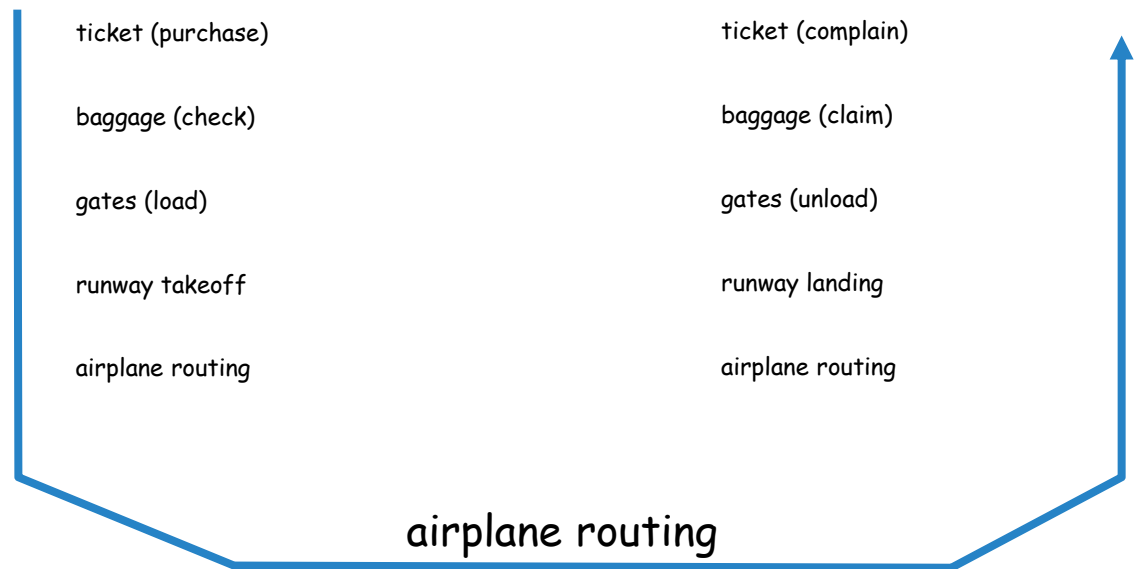
Lot of functions need to be organized

Communicating entities are widely distributed (inherits heterogeneity)

How to manage and enhance ?

- Categorize function groups (services) into disjoint sets
- Assign every group to set of similar entities
- Each entity can perform service to other (upper) entity
- Upper entity a summarized command
- Lower layer performs the details
- This is called the layered model

ORGANIZATION OF AIR TRAVEL



Although this course is about network programming
(and not about networking in general),
an understanding of a complete reversion on network model is essential.

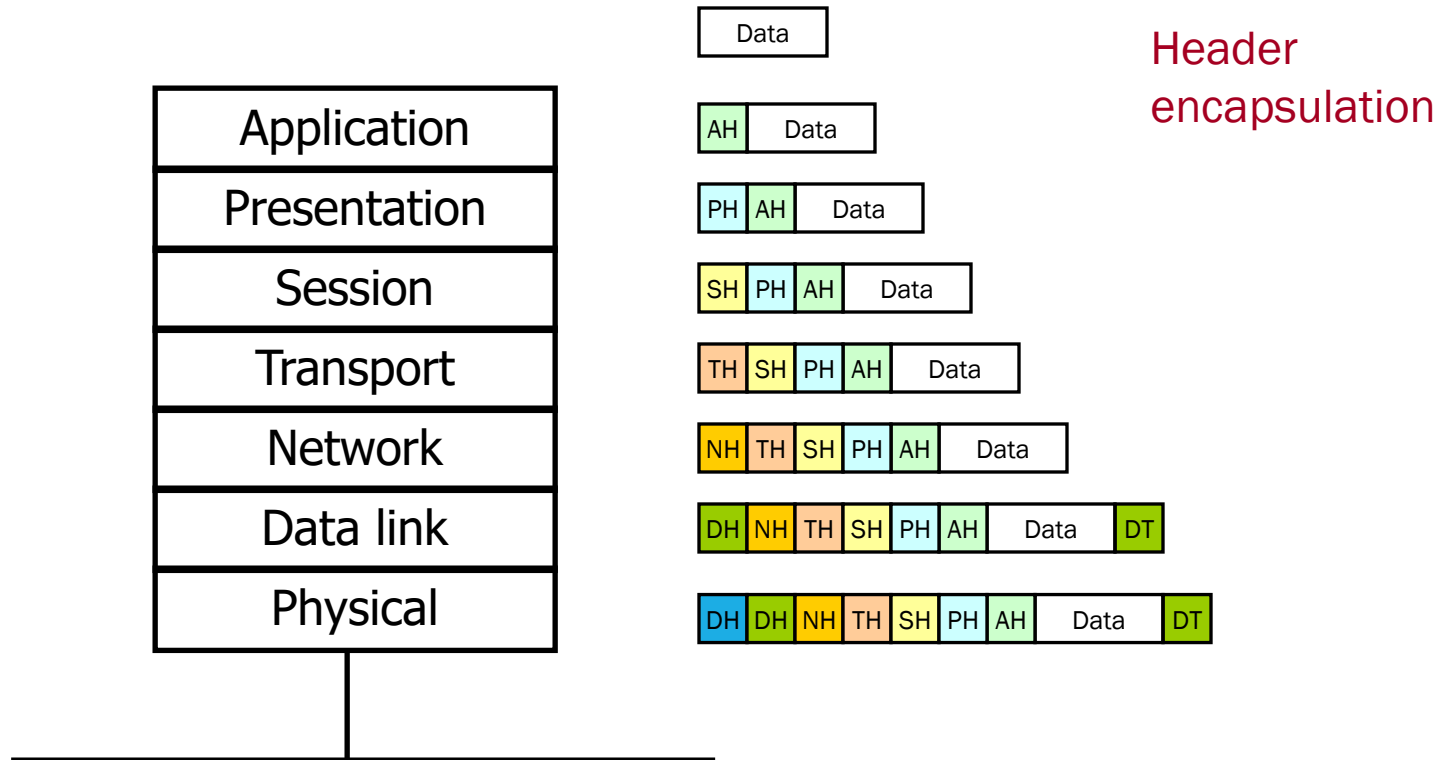
ORGANIZATION OF AIR TRAVEL: **PEER LAYER VIEW**

ticket (purchase)	ticket (complain)
baggage (check)	baggage (claim)
gates (load)	gates (unload)
runway takeoff	runway landing
airplane routing	airplane routing
airplane routing	

Layers: each layer implements a service

- ❑ via its own internal-layer actions
- ❑ relying on services provided by layer below

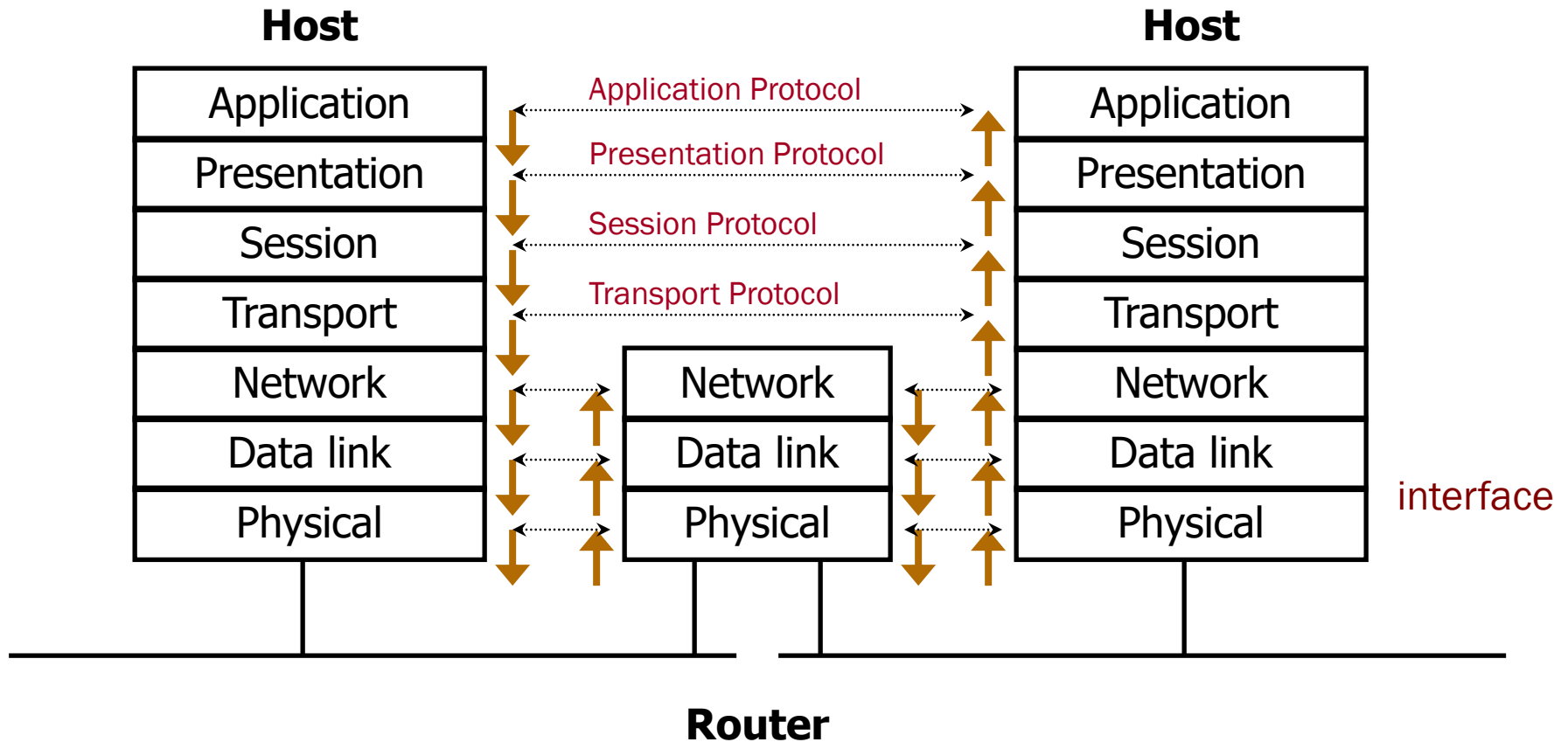
PROTOCOL STACK: ISO OSI MODEL



ISO: the International Standards Organization

OSI: Open Systems Interconnection Reference Model (1984)

COMMUNICATING BETWEEN END HOSTS



PROTOCOLS EXAMPLE

Level Name	Layer Name	Example Protocol
Level 7	Application layer	FTP
Level 6	Presentation layer	XNS Xerox Network Systems
Level 5	Session layer	RPC
Level 4	Transport layer	TCP
Level 3	Network layer	IP
Level 2	Data-Link layer	Ethernet Frames
Level 1	Physical layer	Voltages

INTEROPERABILITY

- Divide a task into pieces and then solve each piece independently (or nearly so).
- Establishing a well-defined **interface** between layers makes porting easier.
- Functions of each layer are **independent** of functions of other layers
 - Thus, each layer is like a module and can be developed independently
- Each layer builds on services provided by lower layers
 - Thus, no need to worry about details of lower layers - *transparent* to this layer
- Major Advantages:
 - Code Reuse
 - Eases maintenance, updating of system

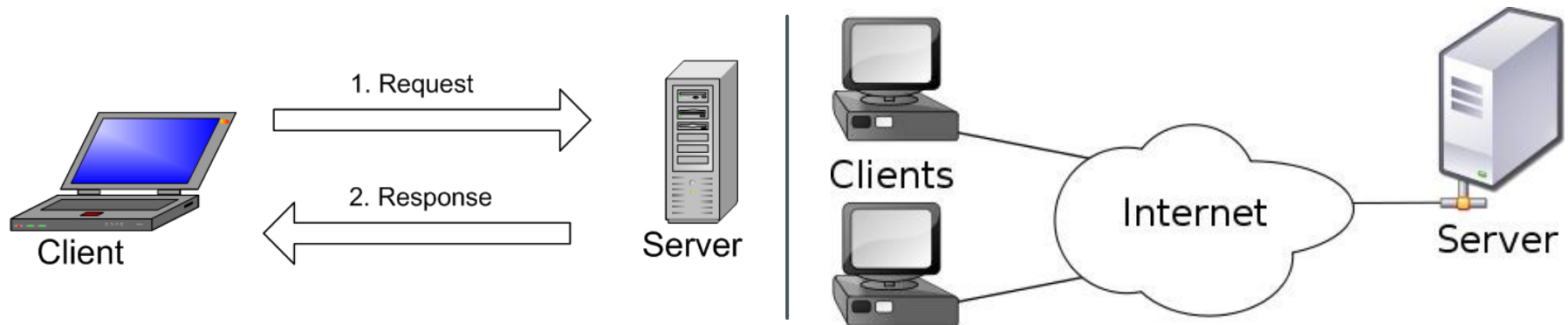
NETWORK PROGRAMS ARCHITECTURE

Client/Server

- **The server hosts, delivers and manages most of the resources and services to be consumed by the client.** This type of architecture has one or more client computers connected to a central server over a network or internet connection

Peer to peer

- **Networking architecture in which each workstation, or node, has the same capabilities and responsibilities**



CLIENT - SERVER

- A server is a process - not a machine !
- A server waits for a request from a client.
- A client is a process that sends a request to an existing server and (usually) waits for a reply.

CLIENT - SERVER EXAMPLES



Server returns the time-of-day.



Server returns a document (FTP server).



Server prints a file for client.



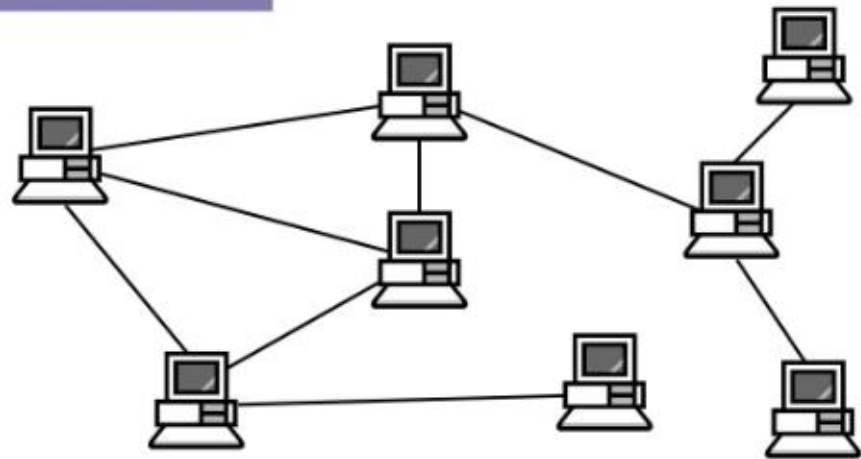
Email



Youtube
(Video Streaming)

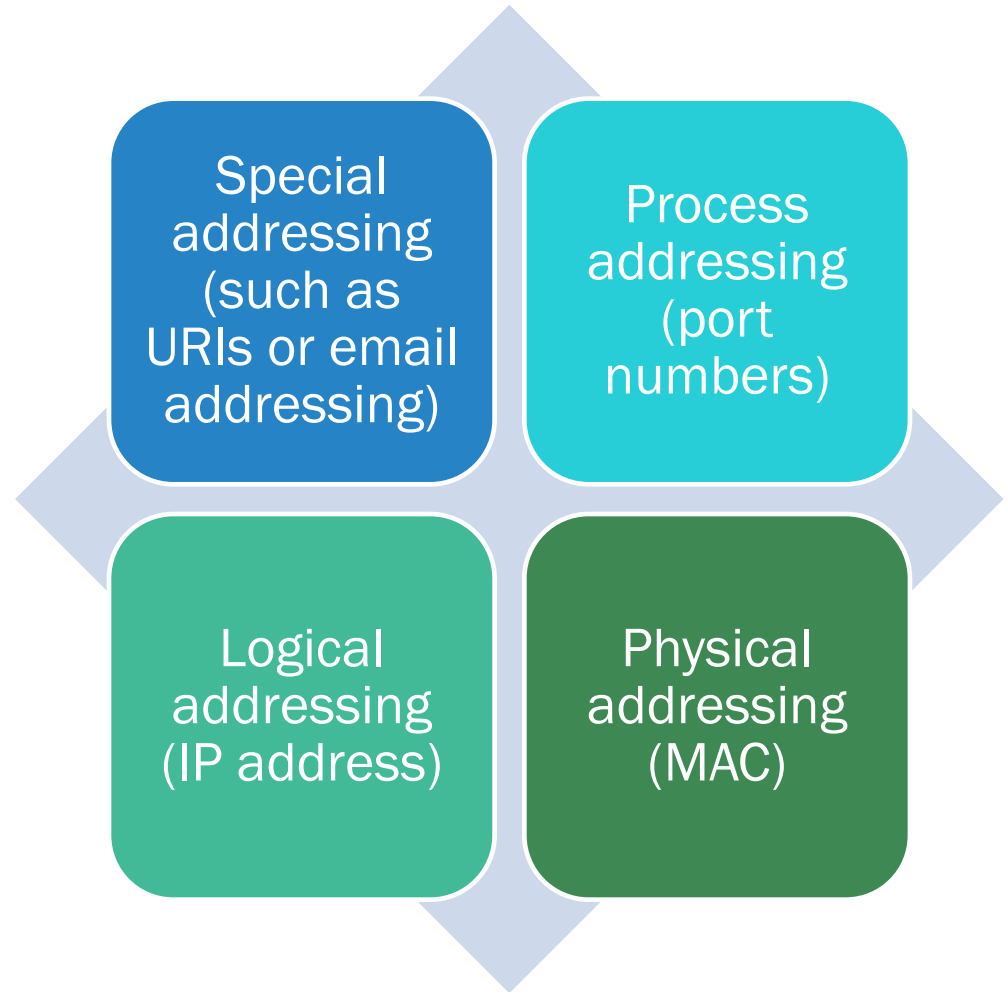
PEER TO PEER

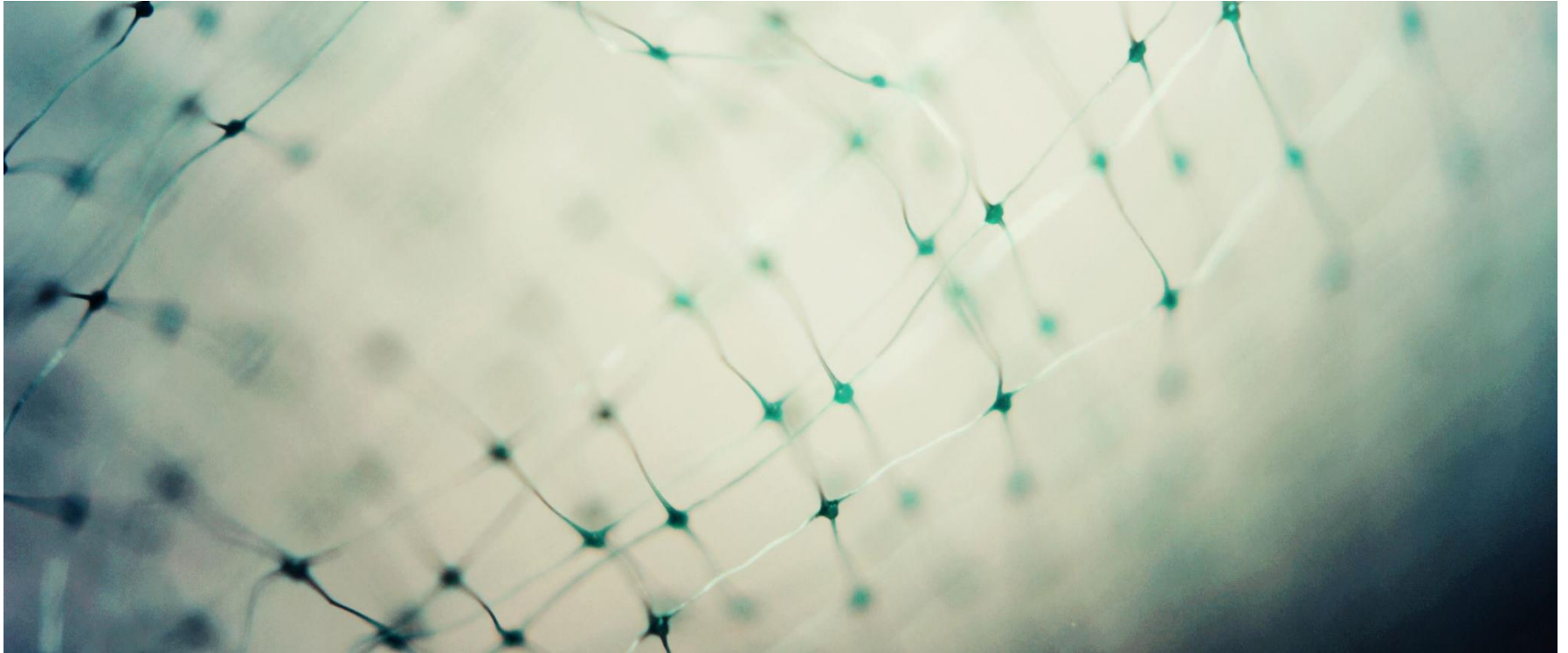
Pure Peer-to-Peer Architecture



- No central server
- Clients connected to one or more peers
- Resilient
- Large #messages

NETWORK ADDRESSING





.NET AND C# FEATURES AND REVISION

PROGRAMS & PROCESSES

- A *program* is an executable file.
- A *process* or *task* is an instance of a program that is being executed.
- A single program can generate multiple processes.

WHY C#.NET

- The Microsoft .NET Framework provides a layered, extensible, and managed implementation of Internet services that can be quickly and easily integrated into your applications.
- Your network applications can build on pluggable protocols to automatically take advantage of new Internet protocols, or they can use a managed implementation of the Windows socket interface to work with the network on the socket level.
- In addition to that you did take a visual C# course before.



WHY C#.NET

The .NET framework provides two namespaces, System.Net and System.Net.Sockets for network programming.

The classes and methods of these namespaces help us to write programs, which can communicate across the network.

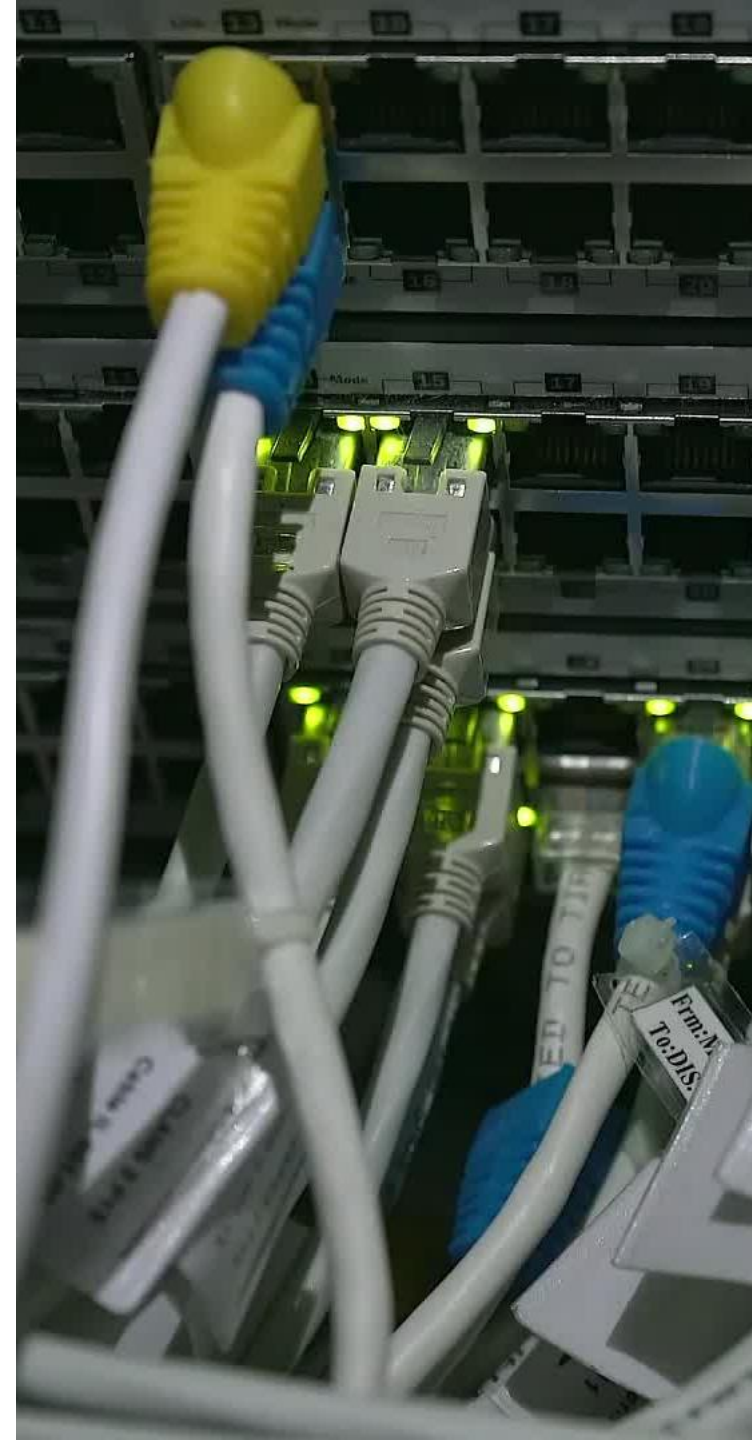
The communication can be either connection oriented or connectionless.

They can also be either stream oriented or datagram based.

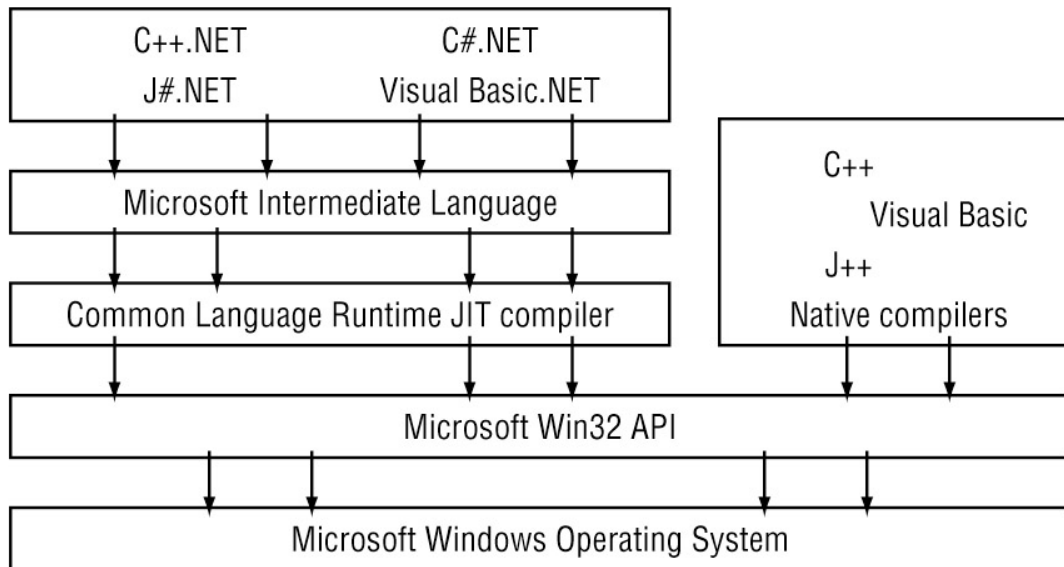
The most widely used protocol is TCP which is used for stream-based communication and UDP is used for data-grams based applications.

LOT OF EASY FEATURES

- There are some other helper classes like `IPEndPoint`, `IPAddress`, `SocketException` etc, which we can use for Network programming.
- The .NET framework supports both synchronous and asynchronous communication between the client and server.
- A synchronous method is operating in blocking mode, in which the method waits until the operation is complete before it returns.
- But an asynchronous method is operating in non-blocking mode, where it returns immediately, possibly before the operation has completed.



.Net Architecture



**.NET MANAGED
CODE
EXECUTION
SCENARIO**

**THE COMMON
LANGUAGE
RUNTIME (CLR)
ENVIRONMENT**




.NET “EXECUTABLE” FILE

- Don't contain ready to run machine code but it has:
 - A *stub* assembly language program to start the CLR compiler
 - The MSIL code of the compiled application

INSTALLING A C# DEVELOPMENT ENVIRONMENT

VISUAL STUDIO 2022 COMMUNITY EDITION



Visual Studio 2022 | 

The most comprehensive IDE for .NET and C++ developers on Windows for building web, cloud, desktop, mobile apps, services and games.

Preview

Get early access to latest features not yet in the main release

[Learn more →](#)

Community

Powerful IDE, free for students, open-source contributors, and individuals

[Free download](#)

Professional

Professional IDE best suited to small teams

[Free trial](#)

Enterprise

Scalable, end-to-end solution for teams of any size

[Free trial](#)



LAB1

- Please do Sheet 1 with your TA.



C# STREAMS

- The Stream Concept
- System.IO namespace
 - Identify the stream classes
 - Types of streams
 - File streams
 - Memory streams

THE STREAM CONCEPT

- A stream is a flow of data from a program to a backing store, or from a backing store to a program.
- The program can either write to a stream or read from a stream.





STREAMS AND STREAM PROCESSING

- Reading from or writing to files in secondary memory (disk)
- Reading from or writing to primary memory (RAM)
- Connection to the Internet
- Socket connection between two programs

SYSTEM.IO NAMESPACE

- Contains types that allow reading and writing to files and data streams, and types that provide basic file and directory support.

CLASSES	
<u>BinaryReader</u>	Reads primitive data types as binary values in a specific encoding.
<u>BinaryWriter</u>	Writes primitive types in binary to a stream and supports writing strings in a specific encoding.
<u>Directory</u>	Exposes static methods for creating, moving, and enumerating through directories and subdirectories. This class cannot be inherited.
<u>File</u>	Provides static methods for the creation, copying, deletion, moving, and opening of a single file, and aids in the creation of <u>FileStream</u> objects.
<u>FileStream</u>	Provides a <u>Stream</u> for a file, supporting both synchronous and asynchronous read and write operations.
<u>FileStreamOptions</u>	Defines a variety of configuration options for <u>FileStream</u> .

<u>IOException</u>	The exception that is thrown when an I/O error occurs.
<u>MemoryStream</u>	Creates a stream whose backing store is memory.
<u>Path</u>	Performs operations on <u>String</u> instances that contain file or directory path information. These operations are performed in a cross-platform manner.
<u>Stream</u>	Provides a generic view of a sequence of bytes. This is an abstract class.
<u>StreamReader</u>	Implements a <u>TextReader</u> that reads characters from a byte stream in a particular encoding.
<u>StreamWriter</u>	Implements a <u>TextWriter</u> for writing characters to a stream in a particular encoding.
<u>StringReader</u>	Implements a <u>TextReader</u> that reads from a string.
<u>StringWriter</u>	Implements a <u>TextWriter</u> for writing information to a string. The information is stored in an underlying <u>StringBuilder</u> .
<u>TextReader</u>	Represents a reader that can read a sequential series of characters.
<u>TextWriter</u>	Represents a writer that can write a sequential series of characters. This class is abstract.

The Abstract class *Stream* in C#

- The abstract class *Stream* is the most general stream class in C#
- Provides a generic view on data sources and data destinations
- Isolates the programmer from operating system details
- Offers reading and writing of raw, binary data
- Chunked and sequenced as bytes
- No char encoding is involved
- **Synchronous** reading and writing
 - Read and Write transfer byte arrays
 - A subclass must implement the operations Read and Write
- **Asynchronous** reading and writing -
 - BeginRead and EndRead
 - BeginWrite and EndWrite
 - Rely on Read and Write
- A stream can be queried for its capabilities
 - CanRead, CanWrite, CanSeek

THE MOST IMPORTANT MEMBERS IN CLASS STREAM

```
int Read (byte[] buf, int pos, int len)
```

```
int ReadByte()
```

```
void Write (byte[] buf, int pos, int len)
```

```
void WriteByte(byte b)
```

```
bool CanRead
```

```
bool CanWrite
```

```
bool CanSeek
```

```
long Length
```

```
void Seek (long offset, SeekOrigin org)
```

```
void Flush ()
```

```
void Close()
```

NON-ABSTRACT SUBCLASSES OF *STREAM*

System.IO.FileStream

- Provides a stream backed by a file from the operating system

System.IO.BufferedStream

- Encapsulates buffering around another stream

System.IO.MemoryStream

- Provides a stream backed by RAM memory

System.Net.Sockets.NetworkStream (this is ours !!)

- Encapsulates a socket connection as a stream

System.IO.Compression.GZipStream

- Provides stream access to compressed data

System.Security.Cryptography.CryptoStream



- Thank you

NETWORK PROGRAMMING

IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

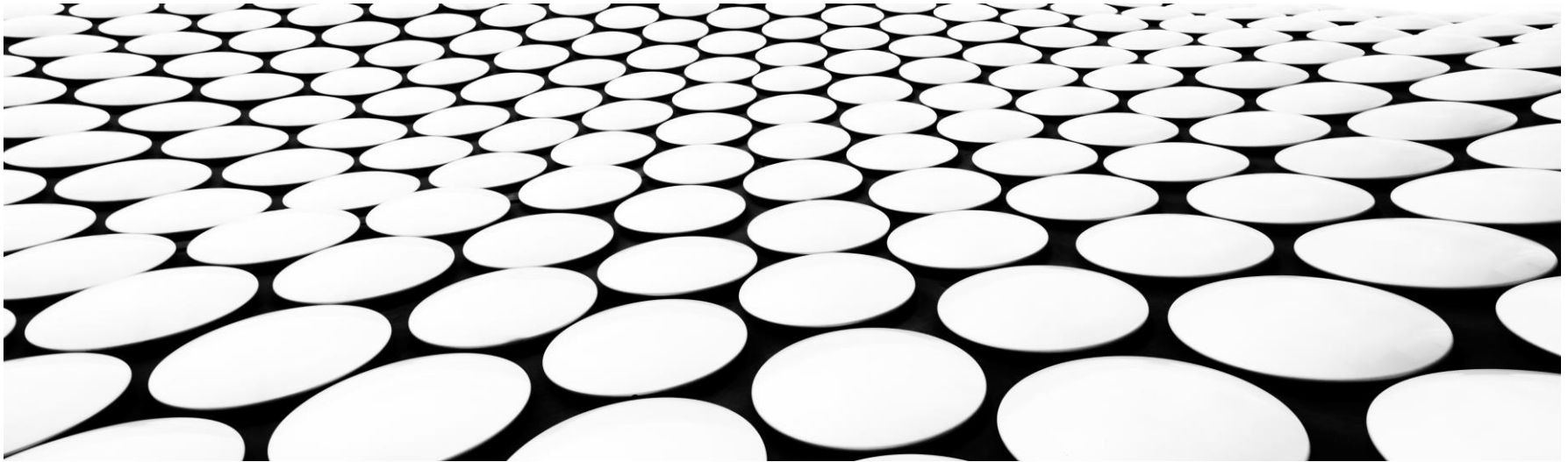
2022



كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

LECTURE 1

PROGRAMMING THE STREAMS

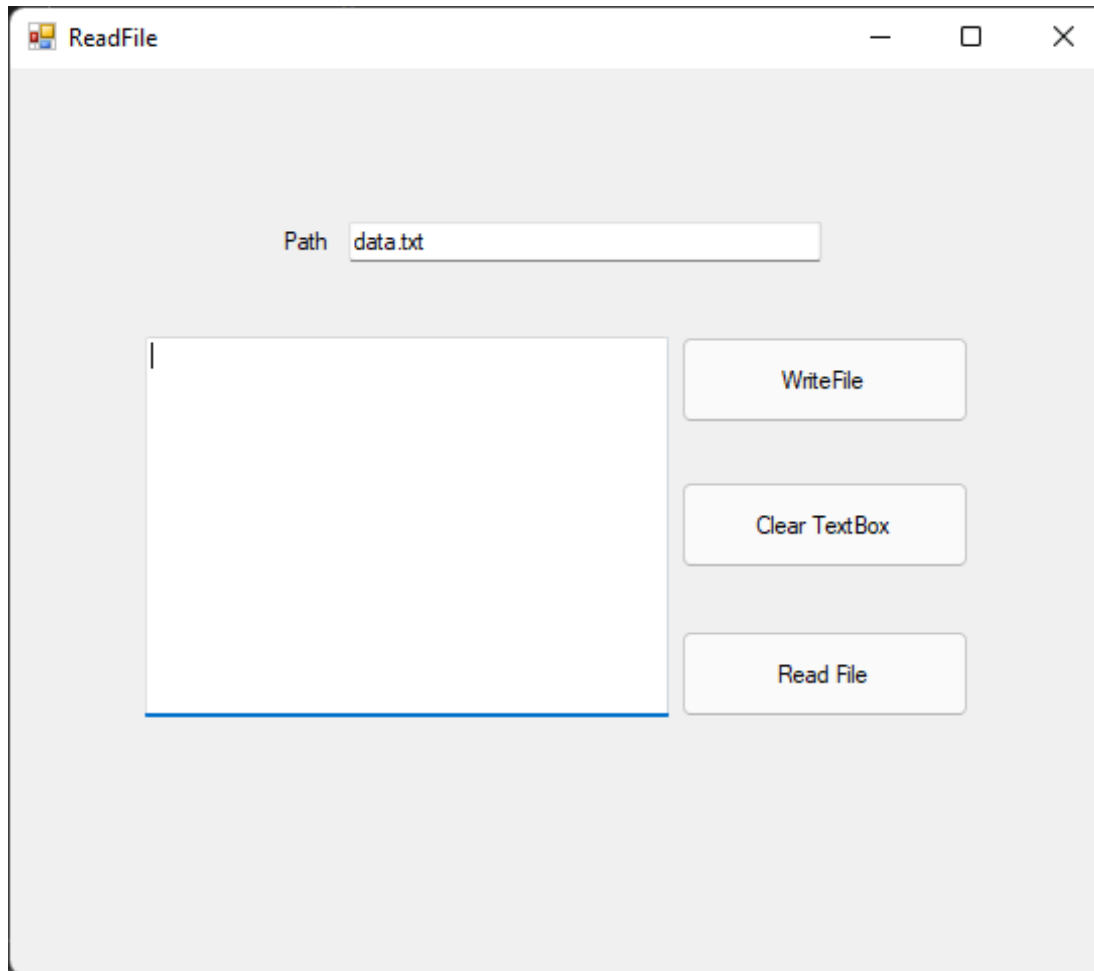




ITS TIME TO CODE !

- Be ready for some examples

EXAMPLE: FILESTREAMS (READ AND WRITE)



Our GUI

WRITE 3 CHARS

```
■ private void button2_Click(object sender, EventArgs e)
■     {
■         Stream s = new FileStream(txtPath.Text, FileMode.Create);
■         s.WriteByte(79); // O 01001111
■         s.WriteByte(79); // O 01001111
■         s.WriteByte(80); // P 01010000
■         s.Close();
■     }
```

READ MORE THAN 3 CHARS

```
■ private void button1_Click(object sender, EventArgs e)
■ {
■     Stream s = new FileStream(txtPath.Text, FileMode.Open);
■     int i, j, k, m, n;
■     i = s.ReadByte(); // O 79 01001111
■     j = s.ReadByte(); // O 79 01001111
■     k = s.ReadByte(); // P 80 01010000

■     m = s.ReadByte(); // -1 EOF
■     n = s.ReadByte(); // -1 EOF

■     txtData.Text = String.Format("{0} {1} {2} {3} {4}", i, j, k, m, n);
■     s.Close();
■ }
```



LAB

- Extend the code to read and write any number of chars, make the same previous GUI functional (problem 1 sheet2)



NOTE

Class **FileStream** offers binary input and output

PRACTICE HINT

- The **using** control structure is helpful when we do IO programming
- Syntax : **using** (*type variable = initializer*) *body*
- Semantics
 - In the scope of using, bind variable to the value of initializer
 - The type must implement the interface **IDisposable**
 - Execute body with the established name binding
 - At the end of body do variable Dispose automatically
 - Usually the Dispose methods in the subclasses of Stream call Close
 - *Why we need this ?*

SYNTAX

```
■ using (Stream s = new FileStream("myFile.txt", FileMode.Create))  
■     {  
■         s.WriteByte(79); // O 01001111  
■         s.WriteByte(79); // O 01001111  
■         s.WriteByte(80); // P 01010000  
■     }
```


WHY USING

- `// The using statement ...`
- `using (type variable = initializer)`
- `body`
- `{type variable = initializer;`
- `try {`
- `body`
- `}`
- `finally {`
- `if (variable != null)`
- `((IDisposable)variable).Dispose();`
- `}`
- `}`

CONSOLE APPLICATION TO COPY FILES

- This example copies the content of a file to another.
- A console application: files names are passed as a command line arguments.
 - **copy-file source-file.txt target-file.txt**
- During your lab, convert it to WinForm APP, problem 2 sheet 2
- Make your GUI Attractive, Plz

```
using System;
using System.IO;
public class CopyApp
{
    public static void Main(string[] args) {
        FileCopy(args[0], args[1]);
    }
    public static void FileCopy(string fromFile, string toFile) {
        try {
            using (FileStream fromStream = new FileStream(fromFile, FileMode.Open))
            {
                using (FileStream toStream = new FileStream(toFile, FileMode.Create))
                {
                    int c;
                    do
                    {
                        c = fromStream.ReadByte();
                        if (c != -1) toStream.WriteByte((byte)c);
                    } while (c != -1);
                }
            }
        }
        catch (FileNotFoundException e)
        {
            Console.WriteLine("File {0} not found: ", e.FileName);
            throw;
        }
        catch (Exception)
        {
            Console.WriteLine("Other file copy exception");
            throw;
        }
    }
}
```

ADAPTER CLASSES

- A set of classes provided by FCL to support reading and writing in higher level

	Input	Output
Text	<i>TextReader (base)</i> StreamReader StringReader	<i>TextWriter (base)</i> StreamWriter StringWriter
Binary	BinaryReader	BinaryWriter

ADAPTER CLASSES

- Higher level of abstraction
 - IO of chars and text strings - not just raw bytes
 - IO of values in simple types
- The *TypeReader* and *TypeWriter* classes are not subclasses of the stream classes
 - They are typically built on - and delegates to - a Stream object
 - A Reader or Writer classes has a stream - it is not a stream
 - Reader and Writer classes serve as Stream adapters



THE CLASS ENCODING

- An encoding is a mapping between characters/strings and byte arrays
- An object of class ***System.Text.Encoding*** represents knowledge about a particular character encoding

THE CLASS ENCODING

- `byte[] GetBytes(string)` **Instance method**
- `byte[] GetBytes(char[])` **Instance method**
 - Encodes a string/char array to a byte array relative to the current encoding
- `char[] GetChars(byte[])` **Instance method**
 - Decodes a byte array to a char array relative to the current encoding
- `byte[] Convert(Encoding, Encoding, byte[])` **Static method**
 - Converts a byte array from one encoding (first parameter) to another encoding (second parameter)

IT TIME TO CODE!

- The following example shows how you can use the Encoding class for either converting to and from byte and also convert between different encodings.
- From a unicode string to a byte array in a given encoding (*GetBytes - Encoding*).
- From a byte array in one encoding to a byte array in another encoding (**Convert**)
- From a byte array in a given encoding to a char array (in unicode) (*GetChars - Decoding*).
- From a char array (in unicode) to a unicode string encoding. (*via String constructor*)


```
using System;
using System.Text;
/* Adapted from an example provided by Microsoft */
class ConvertExampleClass{
    public static void Main()  {
        string unicodeStr =      // "A æ u å æ ø i æ å"
            "A \u00E6 u \u00E5 \u00E6 \u00F8 i \u00E6 \u00E5";
        // Different encodings.
        Encoding ascii = Encoding.ASCII, unicode = Encoding.Unicode,
            utf8 = Encoding.UTF8,
            isoLatin1 = Encoding.GetEncoding("iso-8859-1");

        // Encodes the characters in a string to a byte array:
        byte[] unicodeBytes = unicode.GetBytes(unicodeStr),
            asciiBytes = ascii.GetBytes(unicodeStr),
            utf8Bytes = utf8.GetBytes(unicodeStr),
            isoLatin1Bytes = utf8.GetBytes(unicodeStr);

        // Convert from byte array in unicode to byte array in utf8:
        byte[] utf8BytesFromUnicode =
            Encoding.Convert(unicode, utf8, unicodeBytes);
```

```
// Convert from byte array in utf8 to byte array in ascii:
```

```
byte[] asciiBytesFromUtf8 =
```

```
    Encoding.Convert(utf8, ascii, utf8Bytes);
```

```
// Decodes the bytes in byte arrays to a char array:
```

```
char[] utf8Chars = utf8.GetChars(utf8BytesFromUnicode);
```

```
char[] asciiChars = ascii.GetChars(asciiBytesFromUtf8);
```

```
// Convert char[] to string:
```

```
string utf8String = new string(utf8Chars),
```

```
    asciiString = new String(asciiChars);
```

```
// Display the strings created before and after the conversion.
```

```
Console.WriteLine("Original string: {0}", unicodeStr);
```

```
Console.WriteLine("String via UTF-8: {0}", utf8String);
```

```
Console.WriteLine("ASCII converted string: {0}", asciiString);
```

```
}
```

```
}
```

Microsoft Visual Studio Debug Console

Original string: A æ u å æ o i æ å

String via UTF-8: A æ u å æ o i æ å

ASCII converted string: A ? u ? ? ? i ? ?

C:\Users\Dr. Ali\source\repos\WindowsFormsApp1\Encod

.

Press any key to close this window . . .

OUTPUT, JUSTIFY THE “?”

THE CLASS TEXTWRITER

- Class `TextWriter` supports writing of characters and strings via a chosen encoding
- It is also possible to write textual representations of simple types with use of `TextWriter`.
- Subclasses of the abstract `TextWriter` class
 - `StreamWriter`: For character output in a particular character encoding
 - `StringWriter`: For stream access to a string
- Plz code the related problem in Sheet 2

MEMBERS IN CLASS STREAMWRITER

- 7 overloaded constructors
 - Parameters involved: File name, stream, encoding, buffer size
 - `StreamWriter (String)`
 - `StreamWriter (Stream)`
 - `StreamWriter (Stream, Encoding)`
 - *others*
- 17/18 overloaded Write / WriteLine operations
 - Chars, strings, simple types. Formatted output
- Encoding
 - A property that gets the encoding used for this `TextWriter`
- NewLine
 - A property that gets/sets the applied newline string of this `TextWriter`
- *others*



ADAPTER CLASSES CONT.

- Class `TextReader` supports reading of characters via a chosen encoding
- Class `TextReader` does not have methods to read textual representation of simple types
- Subclasses of the abstract `TextReader` class
 - `StreamReader`: For character input in a particular character encoding
 - `StringReader`: For stream access to a string - (discussed later in the lecture)



PROGRAM

- Reading back the text strings encoded in three different ways, with `StreamReader`.
- In the last half part of the program, the binary contents of the three files are read and reported.



SELF CODE STUDY

- Plz download the cs file from my drive and run it
- plz download and run me !

StreamReader MEMBERS

- 10 StreamReader constructors
 - Similar to the StreamWriter constructors
 - **StreamReader(String)**
 - **StreamReader(Stream)**
 - **StreamReader(Stream, bool)**
 - **StreamReader(Stream, Encoding)**
 - *others*
- **int Read()** Reads a single character. Returns -1 if at end of file
- **int Read(char[], int, int)** Returns the number of characters read
- **int Peek()**
- **String ReadLine()**
- **String ReadToEnd()**
- **CurrentEncoding**
 - A property that gets the encoding of this StreamReader



THE BINARYWRITER, AND BINARYREADER CLASSES

- Class BinaryWriter allows us to write values of simple types in binary format
- Encodings are only relevant if values of type char are written
- Class BinaryReader allows us to read values of simple types in binary format
- Symmetric to BinaryWriter

BinaryWriter **MAIN METHODS AND CONSTRUCTORS**

- Two public constructors
 - `BinaryWriter(Stream)`
 - `BinaryWriter(Stream, Encoding)`
- 18 overloaded Write operations
 - One for each simple type
 - `Write(char)`,
 - `Write(char[])`, and `Write(char[], int, int)` - use Encoding
 - `Write(string)` - use Encoding
 - `Write(byte[])` and `Write(byte[], int, int)`
- `Seek(int offset, SeekOrigin origin)`

BinaryReader MAIN METHODS AND CONSTRUCTORS

- Two public constructors
 - `BinaryReader(Stream)`
 - `BinaryReader(Stream, Encoding)`
- 15 individually name *Readtype* operations
 - `ReadBoolean`, `ReadChar`, `ReadByte`, `ReadDouble`, `ReadDecimal`, `ReadInt16`, ...
- Three overloaded `Read` operations
 - `Read()` and `Read (char[] buffer, int index, int count)`
read characters - using `Encoding`
 - `Read (bytes[] buffer, int index, int count)` reads bytes



EXAMPLES

EXAMPLE 1 , BinaryWriter

```
using System;
using System.IO;

public class BinaryWriteSimpleTypes{

    public static void Main(){
        string fn = "simple-types.bin";

        using(BinaryWriter bw =
            new BinaryWriter( new FileStream(fn, FileMode.Create))){
            bw.Write(5);    // 4 bytes
            bw.Write(5.5);  // 8 bytes
            bw.Write(5555M); // 16 bytes (decimal dt)
            bw.Write(5==6);  // 1 bytes
        }

        FileInfo fi = new FileInfo(fn);
        Console.WriteLine("Length of {0}: {1}", fn, fi.Length);
    }
}
```

Length of simple-types.bin: 29

EXAMPLE 2 , BinaryReader

```
using System;  
using System.IO;
```

```
public class BinaryReadSimpleTypes{
```

```
    public static void Main(){  
        string fn = "simple-types.bin";
```

```
Integer i: 5
```

```
Double d: 5,5
```

```
Decimal dm: 5555
```

```
Boolean b: False
```

```
        using(BinaryReader br =  
            new BinaryReader(    new FileStream(fn, FileMode.Open))){  
            int i = br.ReadInt32();  
            double d = br.ReadDouble();  
            decimal dm = br.ReadDecimal();  
            bool b = br.ReadBoolean();
```

```
            Console.WriteLine("Integer i: {0}", i);  
            Console.WriteLine("Double d: {0}", d);  
            Console.WriteLine("Decimal dm: {0}", dm);  
            Console.WriteLine("Boolean b: {0}", b);  
        } } }
```



The **File** and **FileInfo** classes

- The class **FileInfo** represents a file
- The class **File** holds static methods for creation, copying, deletion, moving, and opening of files

Example: the FileInfo classes

```
using System;
using System.IO;
public class FileInfoDemo{
    public static void Main(){
        // Setting up file names
        string fileName = "file-info.cs", fileNameCopy = "file-info-copy.cs";

        // Testing file existence
        FileInfo fi = new FileInfo(fileName); // this source file
        Console.WriteLine("{0} does {1} exist", fileName, fi.Exists ? "" : "not");

        // Show file info properties:
        Console.WriteLine("DirectoryName: {0}", fi.DirectoryName);
        Console.WriteLine("FullName: {0}", fi.FullName);
        Console.WriteLine("Extension: {0}", fi.Extension);
        Console.WriteLine("Name: {0}", fi.Name);
        Console.WriteLine("Length: {0}", fi.Length);
        Console.WriteLine("CreationTime: {0}", fi.CreationTime);

        // Copy one file to another
        fi.CopyTo(fileNameCopy);
        FileInfo fiCopy = new FileInfo(fileNameCopy);
```

```
// Does the copy exist?
```

```
Console.WriteLine("{0} does {1} exist", fileNameCopy, fiCopy.Exists ? "" : "not");
```

```
// Delete the copy again
```

```
fiCopy.Delete();
```

```
// Does the copy exist?
```

```
Console.WriteLine("{0} does {1} exist",  
    fileNameCopy, fiCopy.Exists ? "" : "not"); // !!??
```

```
// Create new FileInfo object for the copy
```

```
FileInfo fiCopy1 = new FileInfo(fileNameCopy);
```

```
// Check if the copy exists?
```

```
Console.WriteLine("{0} does {1} exist", fileNameCopy, fiCopy1.Exists ? "" : "not");
```

```
// Achieve a TextReader (StreamReader) from the file info object
```

```
// and echo the first ten lines in the file to standard output
```

```
using(StreamReader sr = fi.OpenText()){  
    for (int i = 1; i <= 10; i++)  
        Console.WriteLine(" " + sr.ReadLine());  
} } }
```

Program: A demonstration of the File class.

```
using System;
using System.IO;
public class FileDemo
{
    public static void Main()
    {
        // Setup file names
        string fileName = "binarystreams.exe", // this source file
        fileNameCopy = "fileCopy.cs";

        // Does this source file exist?
        Console.WriteLine("{0} does {1} exist",
            fileName, File.Exists(fileName) ? "" : "not");


        // Copy this source file
        File.Copy(fileName, fileNameCopy);

        // Does the copy exist?
        Console.WriteLine("{0} does {1} exist", fileNameCopy,
            File.Exists(fileNameCopy) ? "" : "not");

        // Delete the copy again
        Console.WriteLine("Deleting {0}", fileNameCopy);
        File.Delete(fileNameCopy);

        // Does the deleted file exist
        Console.WriteLine("{0} does {1} exist", fileNameCopy,
            File.Exists(fileNameCopy) ? "" : "not");

        // Read all lines in source file and echo
        // one of them to the console
        string[] lines = File.ReadAllLines(fileName);
        Console.WriteLine("Line {0}: {1}", 6, lines[6]);    }
```

- 
- The class **DirectoryInfo** represents a directory
 - The class **Directory** holds static methods for creating, moving, and enumerating through directories

Program: A demonstration of the DirectoryInfo class.

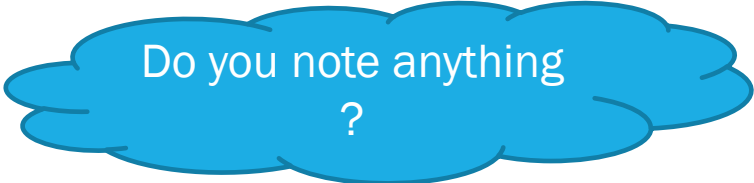
```
using System;
using System.IO;
public class DirectoryInfoDemo
{
    public static void Main()
    {
        string fileName = "directory-info.cs"; // The current source file

        // Get the DirectoryInfo of the current directory
        // from the FileInfo of the current source file
        FileInfo fi = new FileInfo(fileName); // This source file
        DirectoryInfo di = fi.Directory;

        Console.WriteLine("File {0} is in directory \n {1}", fi, di);

        // Get the files and directories in the parent directory.
        FileInfo[] files = di.Parent.GetFiles();
        DirectoryInfo[] dirs = di.Parent.GetDirectories();

        // Show the name of files and directories on the console
        Console.WriteLine("\nListing directory {0}:", di.Parent.Name);
        foreach (DirectoryInfo d in dirs)
            Console.WriteLine(d.Name);
        foreach (FileInfo f in files)
            Console.WriteLine(f.Name);
    }
}
```



Do you note anything
?

Program: A demonstration of the Directory class - similar to the DirectoryInfo demo program.

```
using System;
using System.IO;
public class DirectoryDemo
{
    public static void Main()
    {
        string fileName = "binarystreams.exe";    // The current source file
        FileInfo fi = new FileInfo(fileName);    // This source file

        string thisFile = fi.FullName,
            thisDir = Directory.GetParent(thisFile).FullName,
            parentDir = Directory.GetParent(thisDir).FullName;

        Console.WriteLine("This file: {0}", thisFile);
        Console.WriteLine("This Directory: {0}", thisDir);
        Console.WriteLine("Parent directory: {0}", parentDir);

        string[] files = Directory.GetFiles(parentDir);
        string[] dirs = Directory.GetDirectories(parentDir);

        Console.WriteLine("\nListing directory {0}:", parentDir);
        foreach (string d in dirs)
            Console.WriteLine(d);
        foreach (string f in files)
            Console.WriteLine(f);    } }
```

Output in the next Slide



Output

This file: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0\binarystreams.exe

This Directory: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0

Parent directory: C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug

Listing directory C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug:

C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\Debug\net6.0

NETWORK PROGRAMMING

IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

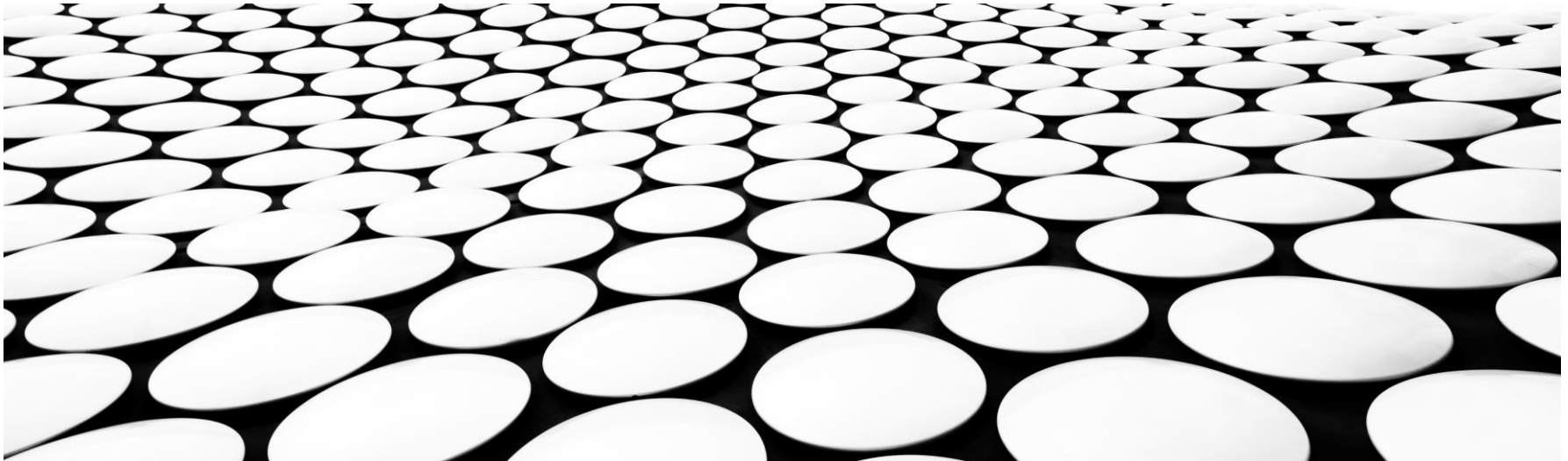
2024



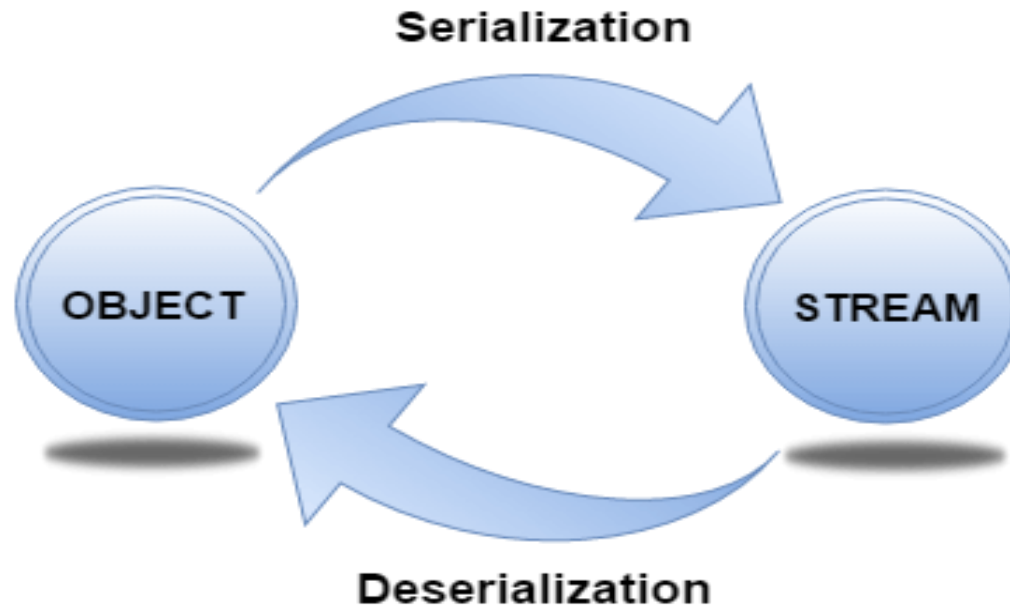
كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

LECTURE 2

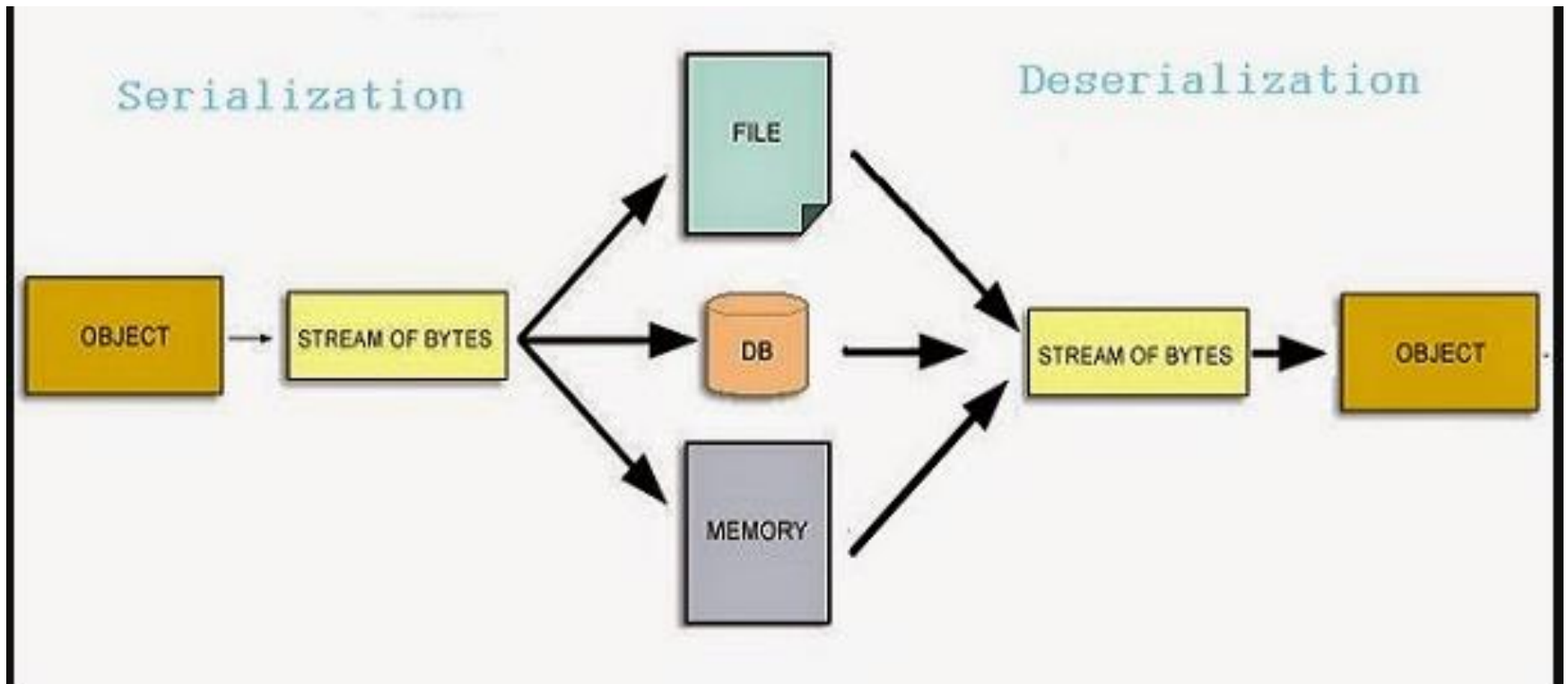
SERIALIZATION



Serialization



What is object serialization ?



Serialization

- **Serialization**
 - Writes an object to a file or any type of stream as a series of bytes (or other formats).
- **Deserialization**
 - Reads a serialized file in order to reestablish or reconstruct the serialized object
- **Serialization and deserialization is supported via classes that implement the *Iformatter* interface:**
 - **BinaryFormatter** and **SoapFormatter**
- **Methods in Iformatter:**
 - **Serialize** and **Deserialize**
- **During this course we will study only **BinaryFormatter** for binary object serialization**

WHY SERIALIZATION ?

- Serialization is the best way to save objects in streams so that we can “send” it any where or even save it immediately.
- Object state can be resorted after deserialization
- Privacy and object state issues !!!
 - What did you think the problem of object state?
 - You can encapsulate objects attributes and protect them, but how we can protect the serialized object ?!

WindowsApplication1 - Microsoft Visual Basic .NET [design] - BookMarks.dat

File Edit View Project Build Debug Tools Window Help

Debug pass

Form2.vb [Design] | Form2.vb | **BookMarks.dat**

00000000	D0 01 00 00 00 FF FF FF	FF 01 00 00 00 00 00 00
00000010	00 0C 02 00 00 00 50 57	69 6E 64 6F 77 73 41 70PWindowsAp
00000020	70 6C 69 63 61 74 69 6F	6E 31 2C 20 56 65 72 73	plication1. Vers
00000030	69 6F 6E 3D 31 2E 30 2E	31 31 32 34 2E 31 33 31	ion=1.0.1124.131
00000040	39 2C 20 43 75 6C 74 75	72 65 3D 6E 65 75 74 72	9. Culture=neutr
00000050	61 6C 2C 20 50 75 62 6C	69 63 4B 65 79 54 6F 6B	al. PublicKeyTok
00000060	65 6E 3D 6E 75 6C 6C 05	01 00 00 00 1C 57 69 6E	en=null.....Win
00000070	64 6F 77 73 41 70 70 6C	69 63 61 74 69 6F 6E 31	dowsApplication1
00000080	2E 42 6F 6F 6B 4D 61 72	6B 04 00 00 00 04 70 55	. Bookmark.....pU
00000090	52 4C 0C 70 44 65 73 63	72 69 70 74 69 6F 6E 08	RL.pDescription.
000000a0	70 4E 65 78 74 55 52 4C	0B 64 61 74 65 43 72 65	pNextURL.dateCre
000000b0	61 74 65 64 01 01 04 00	1C 57 69 6E 64 6F 77 73	ated.....Windows
000000c0	41 70 70 6C 69 63 61 74	69 6F 6E 31 2E 42 6F 6F	Application1. Boo
000000d0	6B 4D 61 72 6B 02 00 00	00 0D 02 00 00 00 06 03	kMark.....
000000e0	00 00 00 15 68 74 74 70	3A 2F 2F 77 77 77 2E 61	...http://vww.a
000000f0	6D 61 7A 6F 6E 2E 63 6F	6D 06 04 00 00 00 13 41	amazon.com.....A
00000100	6D 61 7A 6F 6E 2E 63 6F	6D 20 57 65 62 20 73 69	amazon.com Web si
00000110	74 65 09 05 00 00 00 2E	10 31 E6 3E 54 C5 08 01	te.....1.>T...
00000120	05 00 00 00 01 00 00 00	06 06 00 00 00 16 68 74ht
00000130	74 70 3A 2F 2F 77 77 77	2E 6F 72 65 69 6C 6C 79	tp://vww.oreilly
00000140	2E 63 6F 6D 06 07 00 00	00 11 4F 27 52 65 69 6C	.com.....O'Reil
00000150	6C 79 20 57 65 62 20 53	69 74 65 0A 2E 10 31 E6	ly Web Site...1.
00000160	3E 54 C5 08 0B		>T...

Example of Serialization in C#


- In this example we will serialize and restore an object containing some primitive types
- A controller program will make instance from the class and serialize it to a file and then restore/deserialize it back again into memory



THE USED NAMESPACES

```
using System;  
using System.IO;  
using System.Runtime.Serialization;  
using System.Runtime.Serialization.Formatters.Binary;
```


STEP 1: DEFINING THE CLASSES

```
[Serializable]   
public class ClassToSerialize  
{  
    public int age = 100;  
    public string name = "bipin";  
}
```

STEP 2: ATTEMPTING TO SERIALIZE AND DESERIALIZE



```
public class SerialTest
{
    public void SerializeNow()
    {
        ClassToSerialize c = new ClassToSerialize();

        Stream s = File.Create("temp.dat");
        BinaryFormatter b = new BinaryFormatter();
        b.Serialize(s, c);
        s.Close();
    }

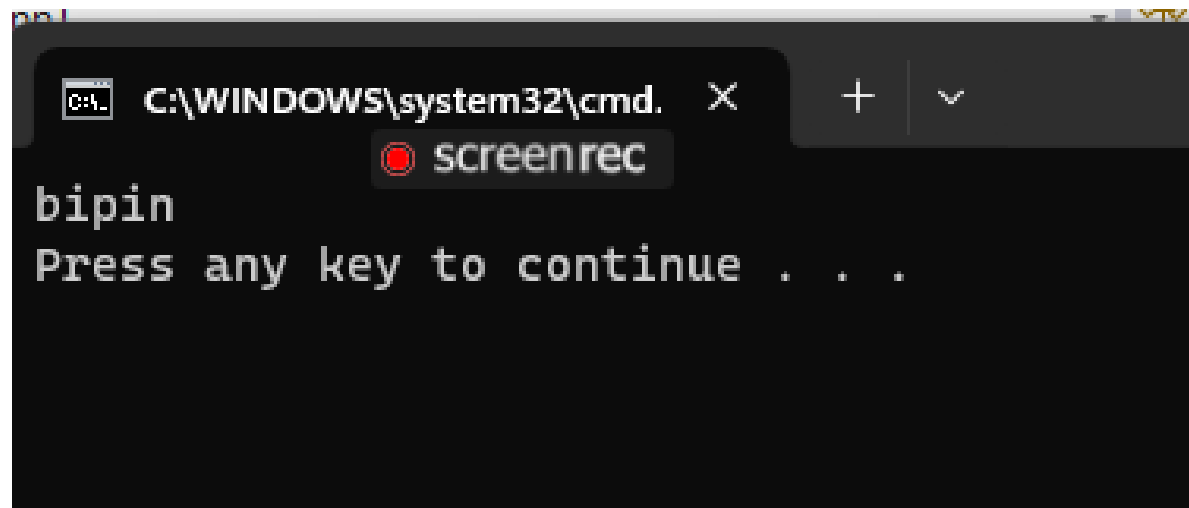
    public void DeSerializeNow()
    {
        ClassToSerialize c = new ClassToSerialize();

        Stream s = File.OpenRead("temp.dat");
        BinaryFormatter b = new BinaryFormatter();
        c = (ClassToSerialize)b.Deserialize(s);
        Console.WriteLine(c.name);
        s.Close();
    }

    public static void Main(string[] s)
    {
        SerialTest st = new SerialTest();
        st.SerializeNow();
        st.DeSerializeNow();
    }
}
```



OUTPUT



```
C:\WINDOWS\system32\cmd. X + v
screenrec
bipin
Press any key to continue . . .
```

Example of Serialization in C#

- In this example we will serialize and restore a *person* object compositing a *date* object.
- We will define class **Date** and **Person**.
- A controller program will make instance from the person class and serialize it to a file and then restore/deserialize it back again into memory



THE USED NAMESPACES

```
using System;  
using System.IO;  
using System.Runtime.Serialization;  
using System.Runtime.Serialization.Formatters.Binary;
```

STEP 1: DEFINING THE CLASSES

[Serializable]

public class Date

{

private ushort year;

private byte month, day;

private DayOfWeek nameOfDay;

public Date(int year, int month, int day)

{

this.year = (ushort)year;

this.month = (byte)month;

this.day = (byte)day;

this.nameOfDay = (new DateTime(year, month, day)).DayOfWeek;

}

public Date(Date d)

{

this.year = d.year; this.month = d.month;

this.day = d.day; this.nameOfDay = d.nameOfDay;

}

STEP 1: DEFINING THE CLASSES

```
public int Year { get { return year; } }
public int Month { get { return month; } }
public int Day { get { return day; } }

// return this minus other, as of usual birthday calculations.
public int YearDiff(Date other)
{
    if (this.Equals(other))
        return 0;
    else return this.year - other.year; ;
}
```

STEP 1: DEFINING THE CLASSES

```
public override bool Equals(Object obj)
{
    if (this.year == ((Date)obj).year &&
        this.month == ((Date)obj).month &&
        this.day == ((Date)obj).day)
        return true;
    else return false;
}
```


STEP 1: DEFINING THE CLASSES

```
public static DateToday
{
    get
    {
        DateTime now = DateTime.Now;
        return new Date(now.Year, now.Month, now.Day);
    }
}

public override string ToString()
{
    return string.Format("{0} {1}.{2}.{3}", nameOfDay, day, month, year);
}
```

STEP 1: DEFINING THE CLASSES

[Serializable]

public class Person

{

private string name;

private int age; // Redundant

private Date dateOfBirth, dateOfDeath;

public Person(string name, Date dateOfBirth)

{

 this.name = name;

 this.dateOfBirth = dateOfBirth;

 this.dateOfDeath = null;

 age = Date.Today.YearDiff(dateOfBirth);

}

public Date DateOfBirth

{

 get { return new Date(dateOfBirth); }

}

public int Age

{

 get { return Alive ? age : dateOfDeath.YearDiff(dateOfBirth); }

}

STEP 1: DEFINING THE CLASSES

```
public bool Alive
```

```
{  
    get { return dateOfDeath == null; }  
}
```

```
public void Died(Date d)
```

```
{  
    dateOfDeath = d;  
}
```

```
public void Update()
```

```
{  
    age = Date.Today.YearDiff(dateOfBirth);  
}
```

```
public override string ToString()
```

```
{  
    return "Person: " + name + " * " + dateOfBirth + (Alive ? " : " + dateOfDeath) +  
        " Age: " + Age;  
}
```

STEP 1: DEFINING THE CLASSES

[OnSerializing()] 

```
internal void OnSerializingMethod(StreamingContext context)
{
    Console.WriteLine("serializing . . .");
}
```

[OnSerialized()] 

```
internal void OnSerializedMethod(StreamingContext context)
{
    Console.WriteLine("Done !");
}
```

[OnDeserializing()] 

```
internal void OnDeserializingMethod(StreamingContext context)
{
    Console.WriteLine("deserializing . . .");
}
```

[OnDeserialized()] 


```
internal void OnDeserializedMethod(StreamingContext context)
{
    Console.WriteLine("obj Ready");
}}
```

STEP 2: ATTEMPTING TO SERIALIZE AND DESERIALIZE

```
class Client
```

```
{
    public static void Main()
    {
        Person p = new Person("Peter", new Date(1936, 5, 11));
        p.Died(new Date(2007, 5, 10));
        Console.WriteLine("{0}", p);
        using (FileStream strm = new FileStream("person.dat", FileMode.Create))
        {
            IFormatter fmt = new BinaryFormatter();
            fmt.Serialize(strm, p);
        }
        // -----
        p = null;
        Console.WriteLine("Reseting person");
        // -----
        using (FileStream strm = new FileStream("person.dat", FileMode.Open))
        {
            IFormatter fmt = new BinaryFormatter();
            p = fmt.Deserialize(strm) as Person;
        }
        Console.WriteLine("{0}", p);  }}

```



OUTPUT



Person: Peter *Monday 11.5.1936 +Thursday 10.5.2007 Age: 85

serializing

Done !

Resetting person

deserializing

obj Ready

Person: Peter *Monday 11.5.1936 +Thursday 10.5.2007 Age: 85

Press any key to close this window . . .



THE “NONSERIALIZED” ATTRIBUTE

- Put [Serializable] on top of the class. For those attributes which you don't want to serialize put [NonSerialized] on them.




A CODE UPDATE TO SEE

```
[NonSerialized]
```

```
    private string name;
```

Guess the output ??



Person: Peter *Monday 11.5.1936 +Thursday 10.5.2007 Age: 71

serializing

Done !

Reseting person

deserializing

obj Ready

Person: *Monday 11.5.1936 +Thursday 10.5.2007 Age: 71

C:\Users\Dr

Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp
1.exe (process 10212) exited with code 0.

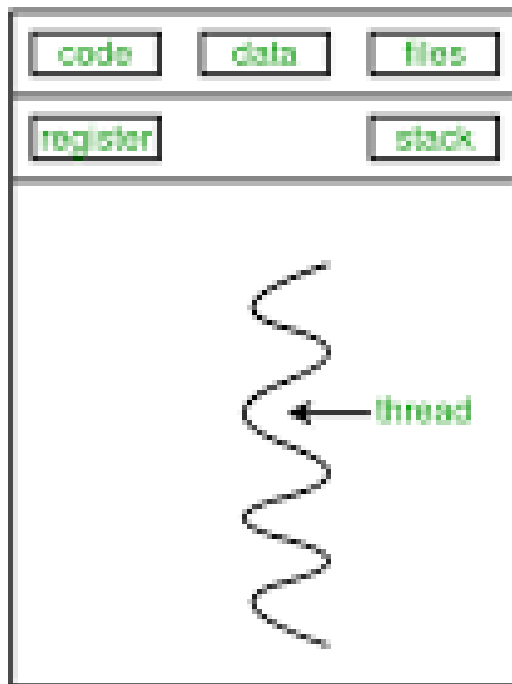
Press any key to close this window . . .



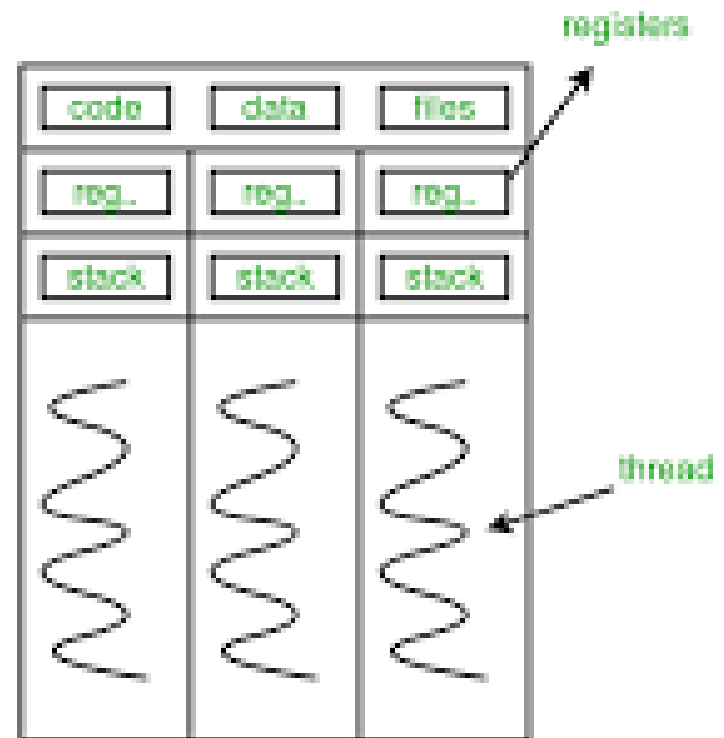
SERIALIZATION BENEFITS

- Serialization allows us to transfer objects through a network by converting it into a byte stream.
- It also helps in preserving the state of the object.
- Deserialization requires less time to create an object than an actual object created from a class. hence serialization saves time.
- One can easily clone the object by serializing it into byte streams and then deserializing it.
- Serialization helps to implement persistence in the program. It helps in storing the object directly in a database in the form of byte streams. This is useful as the data is easily retrievable whenever needed.
- To create a clone of the original object as a backup while working on the main object.
- A set of objects can easily be copied to the system's clipboard and then pasted into the same or another application

Threading



single-threaded process



multithreaded process



WHAT IS A THREAD?

- A thread is nothing more than a process.
- On the computer, a thread is a process moving through time.
- The process performs sets of sequential steps, each step executing a line of code.
- Because the steps are sequential, each step takes a given amount of time.
- The time it takes to complete a series of steps is the sum of the time it takes to perform each programming step.

WHAT ARE MULTITHREADED APPLICATIONS (NETWORK APPLICATIONS)?

- For a long time, most programming applications were single-threaded.
- That means there was only one thread in the entire application.
- You could never do computation A until completing computation B.
- A multithreaded application allows you to run several threads, each thread running in its own process. So theoretically you can run step 1 in one thread and at the same time run step 2 in another thread.
- At the same time you could run step 3 in its own thread, and even step 4 in its own thread
- Theoretically, if all four steps took about the same time, you could finish your program in a quarter of the time it takes to run a single thread (assuming you had a 4 processor machine)
- An Unusual Analogy
 - Human Body

In case of single processor

CONCURRENCY !!





**Can't wait,
lets Code !!**

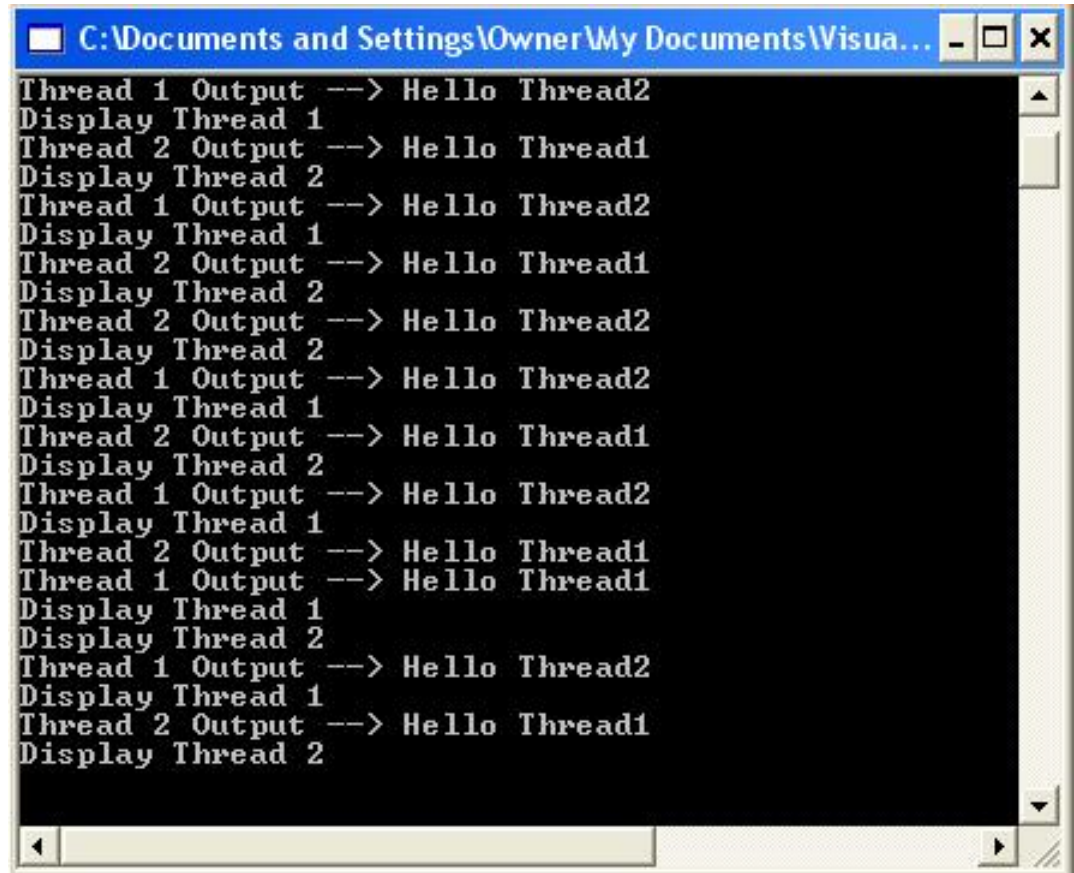
- 
- We are aiming to Create two threads sharing a common variable in memory


```
using System;
using System.IO;
class MThread_APP
{
    // used to indicate which thread we are in
    private static string _threadOutput = "";
    private static bool _stopThreads = false;
    static void DisplayThread1()
    {
        while (_stopThreads == false)
        {
            Console.WriteLine("Display Thread 1");
            // Assign the shared memory to a message about
thread #1
            _threadOutput = "Hello Thread1";
            Thread.Sleep(1000); // simulate a lot of processing
            // tell the user what thread we are in thread #1,
and display shared memory
            Console.WriteLine("Thread 1 Output --> {0}",
_threadOutput);
        }
    }
}
```

```
static void DisplayThread2()
{
    while (_stopThreads == false)
    {
        Console.WriteLine("Display Thread 2");
        // Assign the shared memory to a message about thread #2
        _threadOutput = "Hello Thread2";
        Thread.Sleep(1000); // simulate a lot of processing
        // tell the user we are in thread #2
        Console.WriteLine("Thread 2 Output --> {0}",
            _threadOutput);    }
}

public static void Main()
{
    Thread thread1 = new Thread(new
    ThreadStart(DisplayThread1));
    Thread thread2 = new Thread(new
    ThreadStart(DisplayThread2));
    // start them
    thread1.Start();
    thread2.Start();}}
```

OUTPUT
Any
comment !!



A screenshot of a Windows command prompt window. The title bar reads "C:\Documents and Settings\Owner\My Documents\Visua...". The window contains a list of 24 lines of text, alternating between "Thread 1 Output --> Hello Thread2" and "Display Thread 1", and "Thread 2 Output --> Hello Thread1" and "Display Thread 2". The output is interleaved, showing the execution of two threads simultaneously. The window has a standard Windows XP-style interface with a blue title bar, minimize/maximize/close buttons, and a scrollbar on the right.

```
C:\Documents and Settings\Owner\My Documents\Visua...
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
Thread 1 Output --> Hello Thread2
Display Thread 1
Thread 2 Output --> Hello Thread1
Display Thread 2
```

EXPLANATION

- The code theoretically is executing the two methods DisplayThread1 and DisplayThread2, simultaneously.
- Each method shares the variable, `_threadOutput`.
- Race condition !
 - Each method shares the variable, `_threadOutput`. So it is possible that `_threadOutput` is assigned a value "Hello Thread1" in thread #1 and displays `_threadOutput`
 - Somewhere in between the time thread #1 assigns it and displays it, thread #2 assigns `_threadOutput` the value "Hello Thread2"
 - Thread#1 assigns and thread#2 displays !!
 - How to organize that !!
 - Use **Lock**

THREAD SAFE CODE

- The best way to avoid race conditions is to write thread-safe code
- Make two instances (don't share)
- Let any one to win the race!
- The way we prevent one thread from affecting the memory of the is called *locking*.
- So any thread trying to modify a field of the class while inside the lock will be *blocked*
- Blocking means that the thread trying to change the variable will sit and wait until the lock is released on the locked thread.
- The thread is released from the lock upon reaching the last bracket in the lock `{ }` construct.

```
using System;
using System.IO;
class MThread_APP
{
    // used to indicate which thread we are in
    private string _threadOutput = "";
    private bool _stopThreads = false;
    public MThread_APP() {
        Thread thread1 = new Thread(new ThreadStart(DisplayThread1));
        Thread thread2 = new Thread(new ThreadStart(DisplayThread2));
        // start them
        thread1.Start();
        thread2.Start();
    }
    void DisplayThread1()
    {
        while (_stopThreads == false) {
            // lock on the current instance of the class for thread #1
            lock (this) {
                Console.WriteLine("Display Thread 1");
                _threadOutput = "Hello Thread1";
                Thread.Sleep(1000); // simulate a lot of processing
            }
            // tell the user what thread we are in thread #1
            Console.WriteLine("Thread 1 Output --> {0}",
                _threadOutput);
        }
    }
}
```

```
void DisplayThread2()    {
    while (_stopThreads == false)    {

// lock on the current instance of the class for thread #2
        lock (this)    {
            Console.WriteLine("Display Thread 2");
            _threadOutput = "Hello Thread2";
            Thread.Sleep(1000); // simulate a lot of
processing
                                // tell the user what thread
we are in thread #1
            Console.WriteLine("Thread 2 Output --> {0}", _threadOutput);
                } // lock released for thread #2 here
        }    }}

class MainClass
{
    public static void Main()
    {
        new MThread_APP();
    }}
}
```



```
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
```




LETS HAVE ANOTHER EXAMPLE

WHAT CAN WE USE TO AVOID RACE CONDITION?

- C# provides great facilities which can be used to manage shared areas
- The `AutoResetEvent` class
- Two main methods
 - `WaitOne()` → for the associated obj, wait here don't proceed executing !
 - `Set()` → for the associated obj, ok you can proceed executing

```
using System;
using System.Threading;
class MThread_APP
{
    // used to indicate which thread we are in
    private string _threadOutput = "";
    private bool _stopThreads = false;
    public MThread_APP()
    {
        Thread thread1 = new Thread(new
ThreadStart(DisplayThread_1));
        Thread thread2 = new Thread(new
ThreadStart(DisplayThread_2));
        // start them
        thread1.Start();
        thread2.Start();
    }
    AutoResetEvent _blockThread1 = new AutoResetEvent(false);
    AutoResetEvent _blockThread2 = new AutoResetEvent(true);
}
```

```
void DisplayThread_1() {
    while (_stopThreads == false) {
        // block thread 1 while the thread 2 is executing
        _blockThread1.WaitOne();
        // Set was called to free the block on thread 1,
        continue executing the code
        Console.WriteLine("Display Thread 1");
        _threadOutput = "Hello Thread 1";
        Thread.Sleep(1000); // simulate a lot of processing

        // tell the user what thread we are in thread #1
        Console.WriteLine("Thread 1 Output --> {0}",
        _threadOutput);

        // finished executing the code in thread 1, so unblock
        thread 2
        _blockThread2.Set();
    }
}
```

```
void DisplayThread_2()
{
    while (_stopThreads == false)
    {
        // block thread 2 while thread 1 is executing
        _blockThread2.WaitOne();

        // Set was called to free the block on thread 2,
        continue executing the code
        Console.WriteLine("Display Thread 2");
        _threadOutput = "Hello Thread 2";
        Thread.Sleep(1000); // simulate a lot of processing

        // tell the user we are in thread #2
        Console.WriteLine("Thread 2 Output --> {0}",
        _threadOutput);


        // finished executing the code in thread 2, so
        unblock thread 1
        _blockThread1.Set();
    }
}
```

OUTPUT, WHY THREAD 2 BEGINS?!



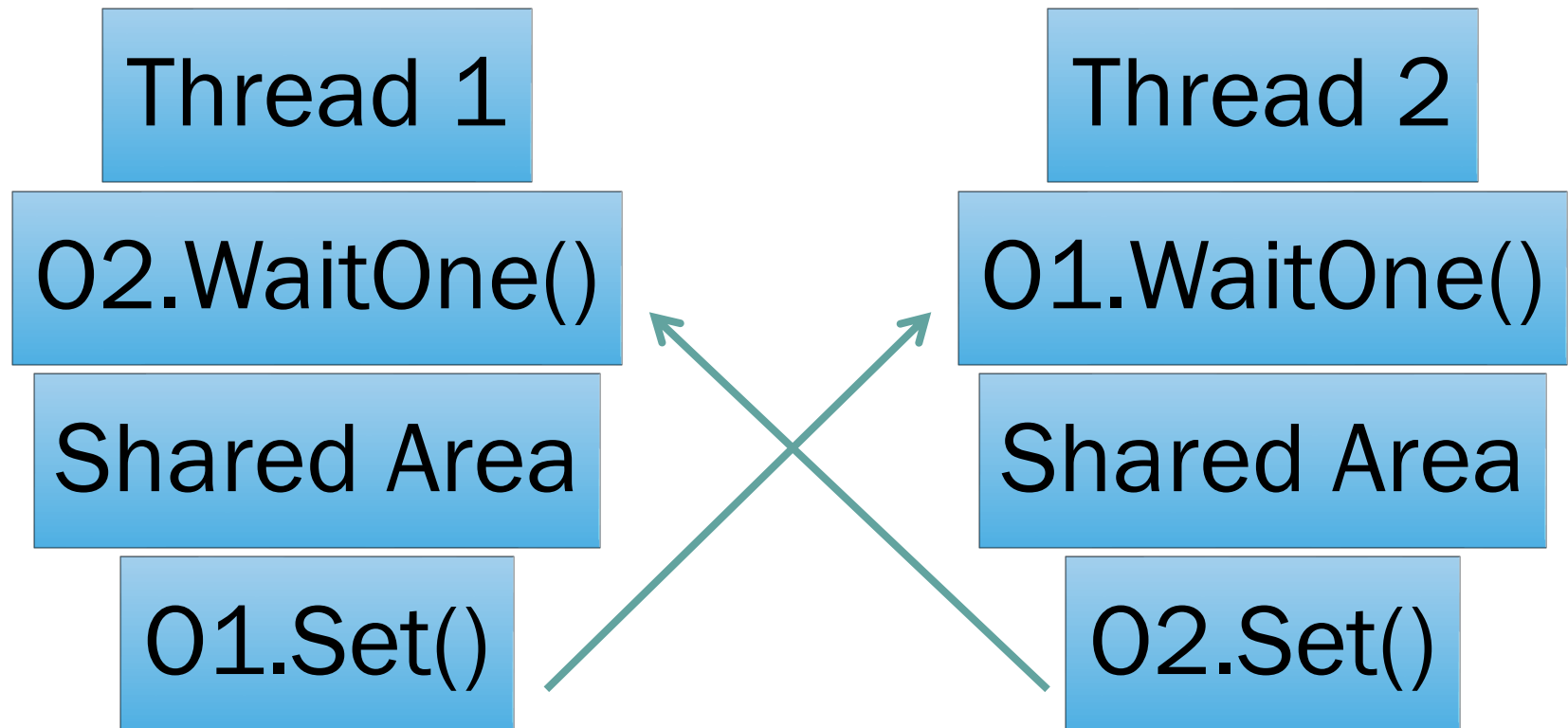
C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe

```
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
Thread 1 Output --> Hello Thread 1
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
Thread 1 Output --> Hello Thread 1
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
Thread 1 Output --> Hello Thread 1
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
```



```
class MainClass
{
    public static void Main()
    {
        new MThread_APP();
    }
}
```

O2.Set Lets O2 to Proceed the Code After O2.WaitOne And Vice Versa



PERFORMANCE USING THREADS

- Keep in mind, creating more threads is not related to processing speed or performance.
- All threads share the same processor and resources a machine has.
- In cases of multiple threads, the thread scheduler with the help of the operating system schedules threads and allocates a time for each thread.
- But in a single processor machine, only one thread can execute at a time.
- The rest of the threads have to wait until the processor becomes available.
- Creating more than a few threads on a single processor machine may create a resource bottleneck if not managed properly.
- **So don't "Thread" Everything**



THANK U

NETWORK PROGRAMMING

IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

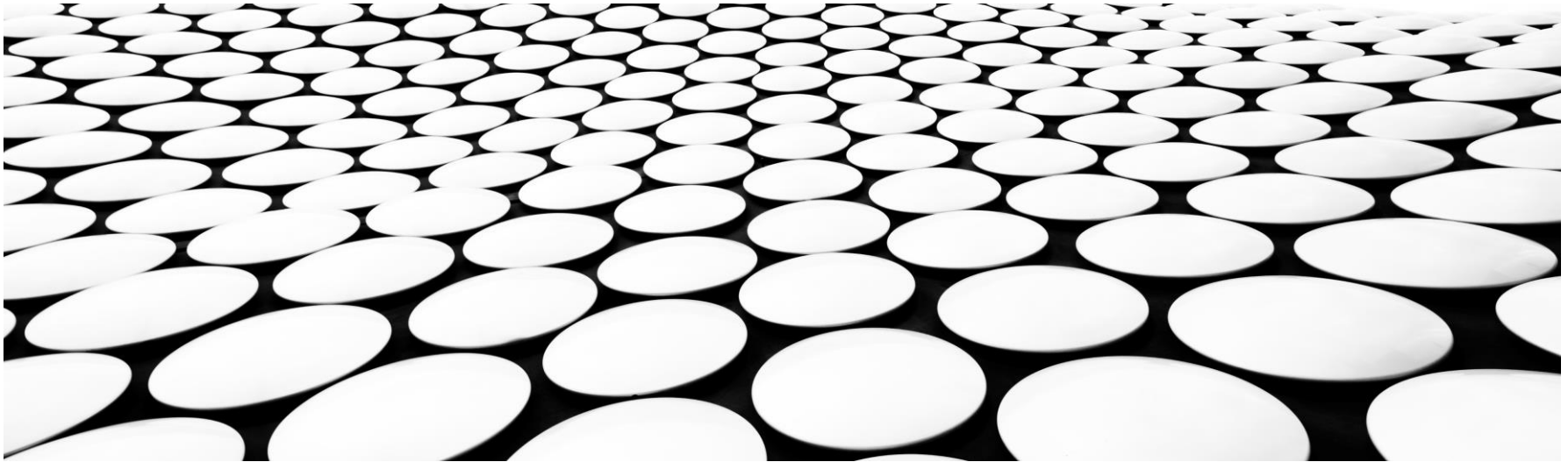
2022



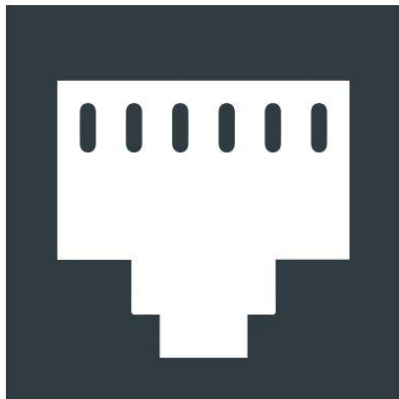
كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

LECTURE 3

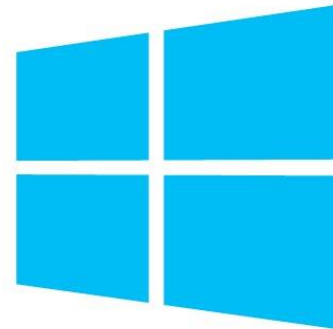
THE BASICS . . .



SOCKETS



SOCKETS





WHAT IS A SOCKET?

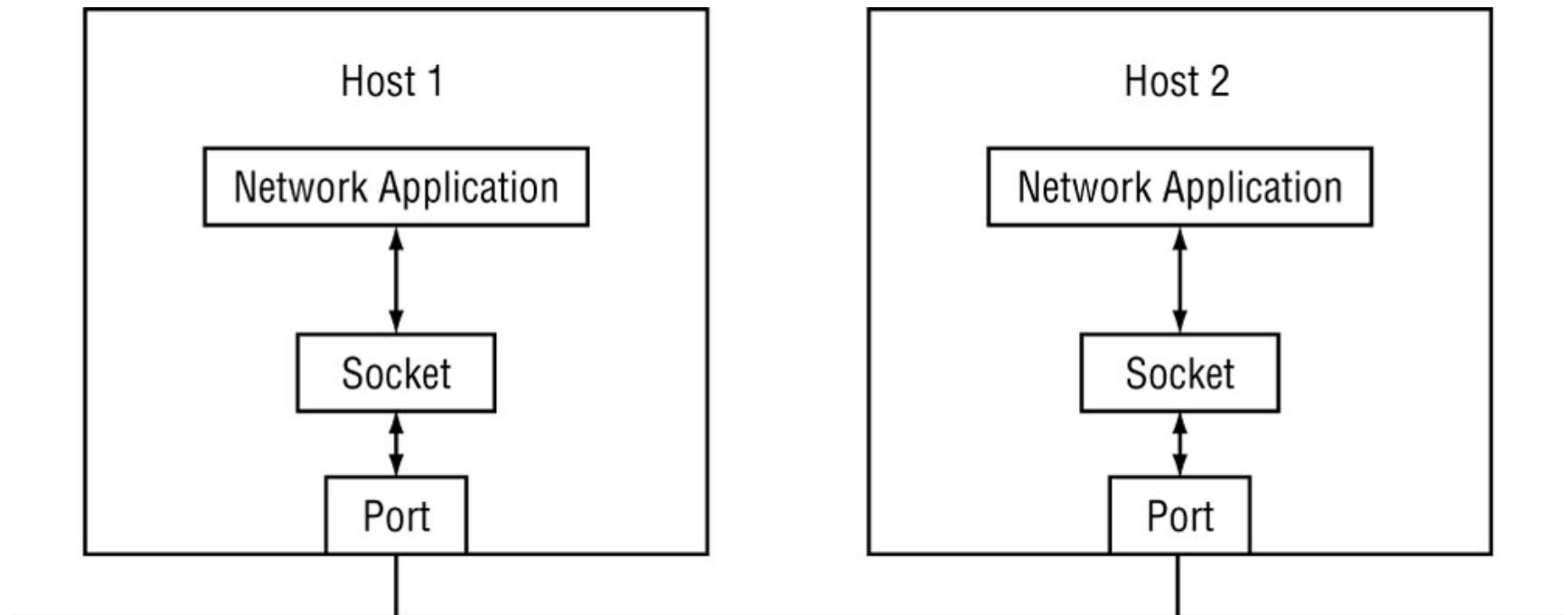
- In socket-based network programming, you do not directly access the network interface device to send and receive packets.
- Instead, an intermediary file descriptor is created to handle the programming interface to the network.
- operating system handles the details of determining which network interface device will be used to send out the data and how.



WHAT DID THE SOCKET DEFINE?

- A specific communication domain, such as a network.
- A specific communication type, such as stream or datagram
- A specific protocol, such as TCP or UDP

THE SOCKET INTERFACE





SOCKET TYPES

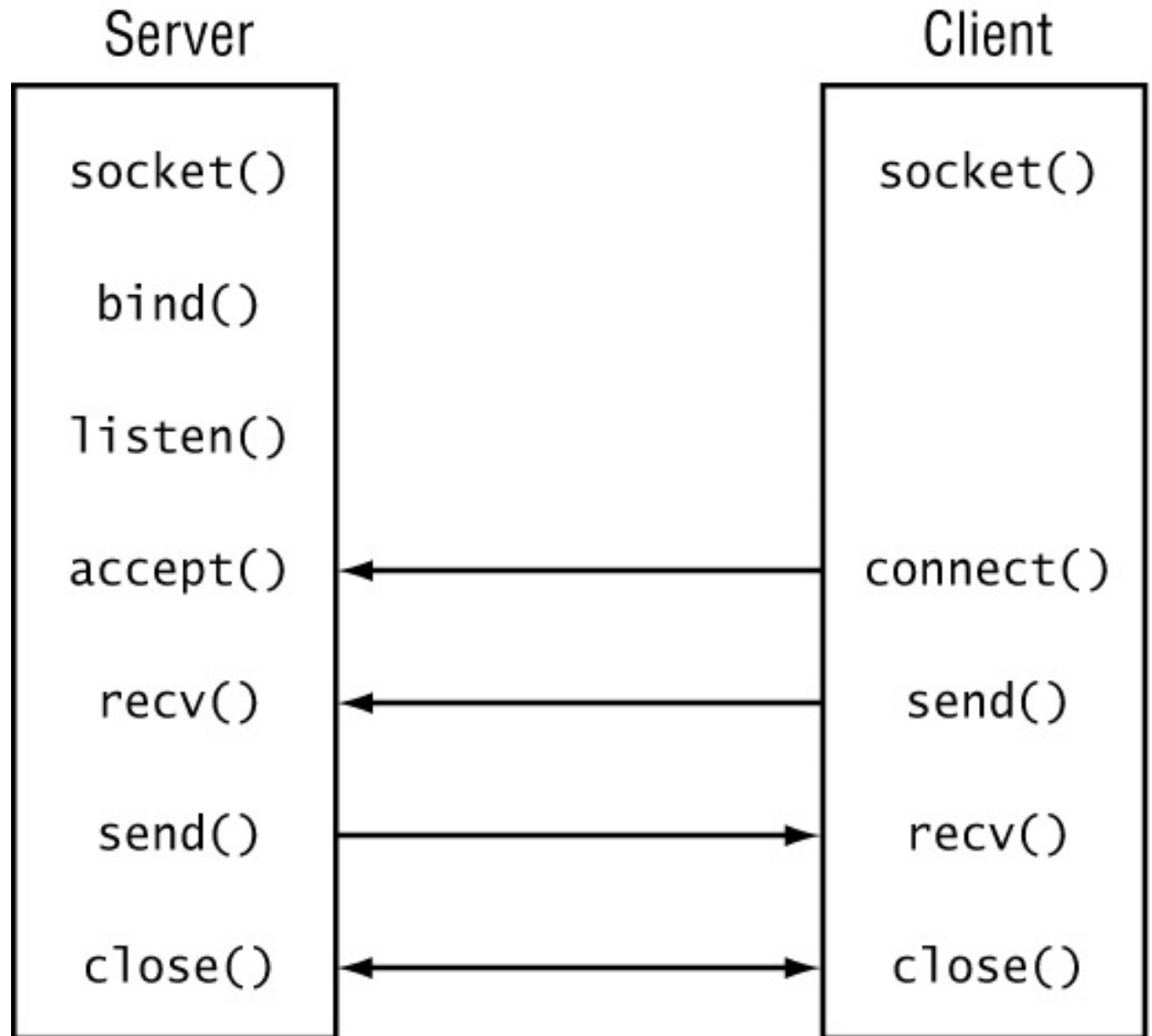
- **Connection-Oriented Sockets**
- **Connectionless Sockets**



CONNECTION-ORIENTED SOCKETS

- In a connection-oriented socket (stream socket) the TCP protocol is used to establish a session (connection) between two IP address endpoints.
- There is a fair amount of overhead involved with establishing the connection, but once it is established, data can be reliably transferred between the devices.
- To create a connection-oriented socket, separate sequences of functions must be used for server programs and for client programs (see the next slide)

STEPS



Server – high level view

Create a socket

Bind the socket

Listen for connections

Accept new client connections

Read/write to client connections

Shutdown connection

CLIENT – HIGH LEVEL VIEW

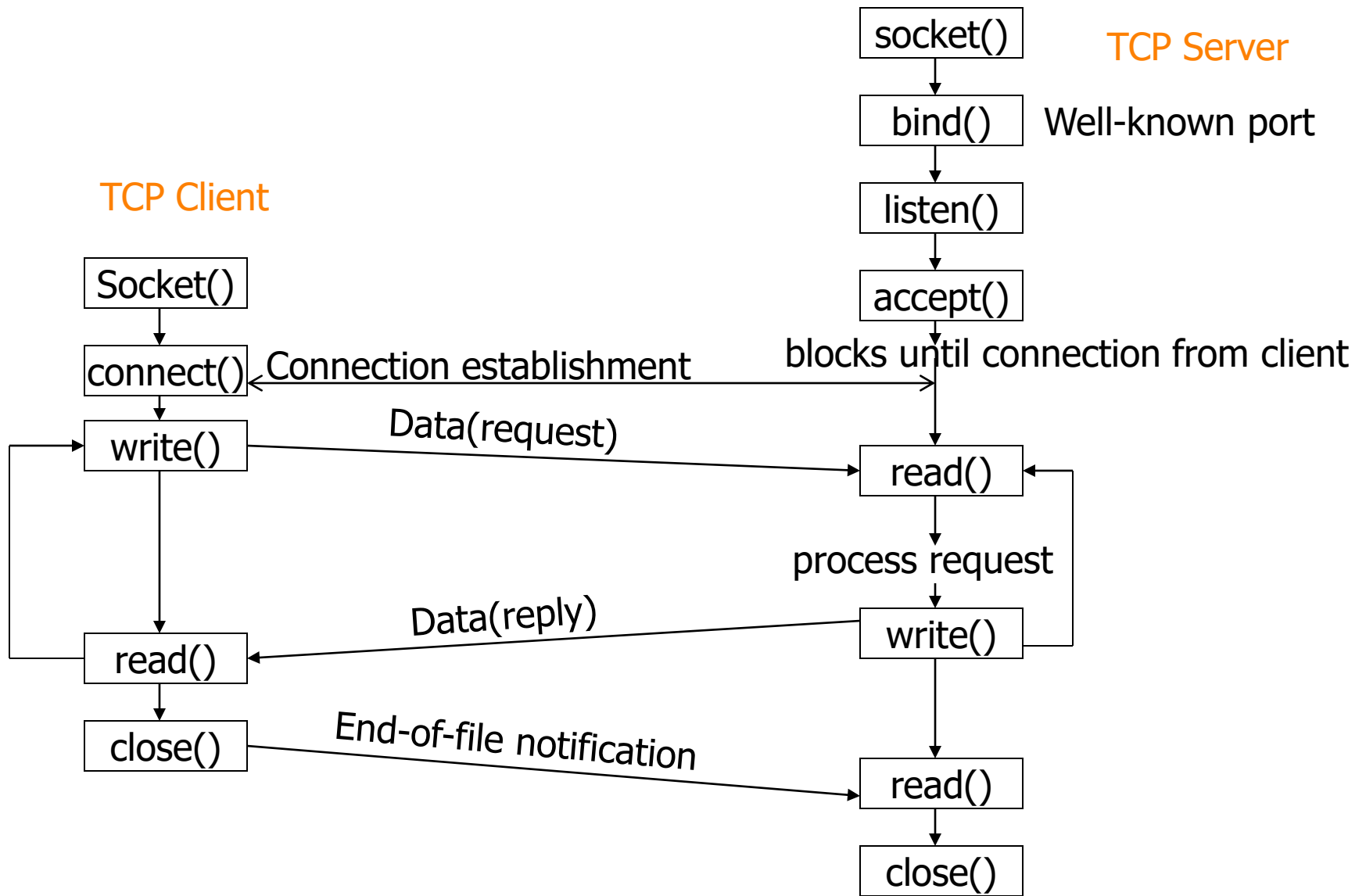
Create a socket

Setup the server address

Connect to the server

Read/write data

Shutdown connection



THE SERVER FUNCTIONS

- For the server program, the created socket must be bound to a local IP address and port number that will be used for the TCP communication.
- After the socket is bound to an address and port, the server program must be ready to accept connections from remote clients. This is a two-step process: first, the program looks for an incoming connection, next it sends and receives data.
- The program must first use the `listen()` function to "listen" to the network for an incoming connection. Next it must use the `accept()` function to accept connection attempts from clients
- After the `listen()` function, the `accept()` function must be called to wait for incoming connections. The format of the `accept()` function
- Once the connection has been accepted, the server can send and receive data from the client using the send and receive



PORTS

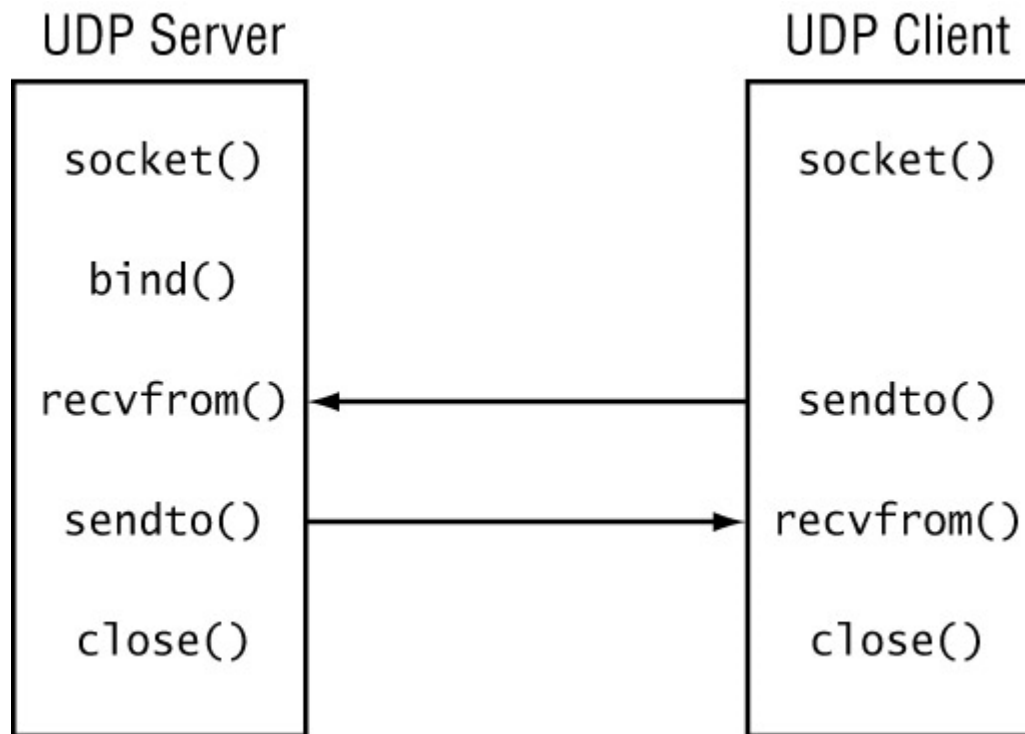
- Well Known Ports are in the range of 0 to 1023 only the operating system or an Administrator of the system can access
- Registered Ports are in the range of 1024 to 49151.
- Dynamic and/or Private Ports are those from 49152 through 65535 and are open for use without restriction



THE CLIENT FUNCTIONS

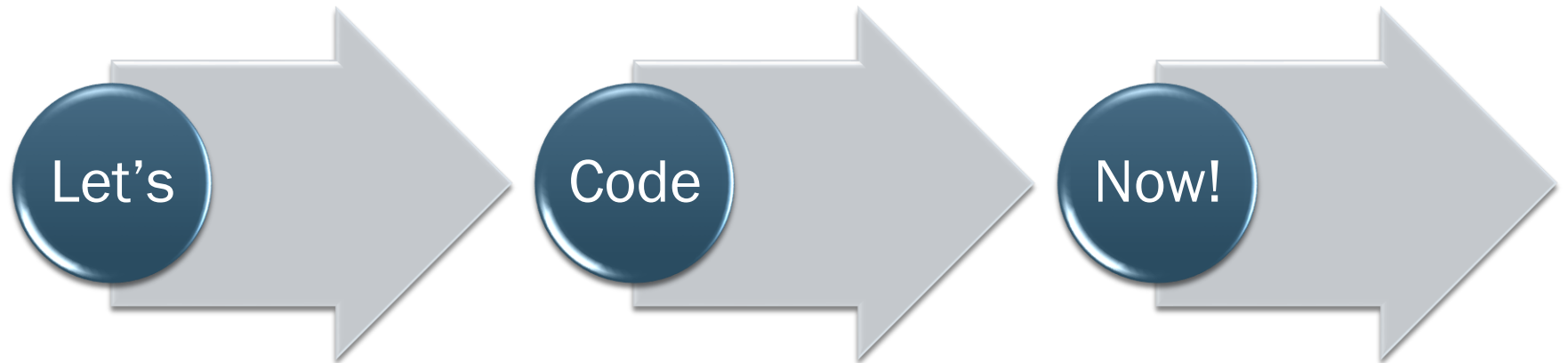
- In a connection-oriented socket, the client must bind to the specific host address and port for the application.
- For client programs, the `connect()` function is used instead of the `listen()` function:
- Once the `connect()` function succeeds, the client is connected to the server and can use the standard `send()` and `receive()` functions to transmit data back and forth with the server.

USING CONNECTIONLESS SOCKETS



USING CONNECTIONLESS SOCKETS

- sockets use the UDP protocol, no connection information is required to be sent between network devices.
- Because of this, it is often difficult to determine which device is acting as a “server”, and which is acting as a “client”.
- If a device is initially waiting for data from a remote device, the socket must be bound to a local address/port pair using the `bind()` function.
- Once this is done the device can send data out from the socket, or receive incoming data from the socket.
- Because the client device does not create a connection to a specific server address, the `connect()` function need not be used for the UDP client program





OUTLINE

- IP Addresses in C#
- Using C# Sockets
- C# Socket Exceptions



IP ADDRESSES IN C#

- .NET defines two classes in the System.Net namespace to handle various types of IP address information:
 - IPAddress
 - IPEndPoint

IPADDRESS

- used to represent a single IP address.
- This value can then be used in the various socket methods to represent the IP address.
- `public IPAddress(long address)`
- `IPAddress newaddress = IPAddress.Parse("192.168.1.1");`

TYPES OF IP ADDRESSES

- **Any** Used to represent any IP address available on the local system
 - Provides an IP address that indicates that the server must listen for client activity on all network interfaces. This field is read-only.
- **Broadcast** Used to represent the IP broadcast address for the local network
- **Loopback** Used to represent the loopback address of the system
- **None** Used to represent no network interface on the system
 - Don't listen for user activity

TRY

```
using System;
using System.Net;
class AddressSample
{
    public static void Main()
    {
        IPAddress test1 = IPAddress.Parse("192.168.1.1");
        IPAddress test2 = IPAddress.Loopback;
        IPAddress test3 = IPAddress.Broadcast;
        IPAddress test4 = IPAddress.Any;
        IPAddress test5 = IPAddress.None;
        IPHostEntry ihe = Dns.GetHostByName(Dns.GetHostName());
        IPAddress myself = ihe.AddressList[0];
    }
}
```

```
if (IPAddress.IsLoopback(test2))
    Console.WriteLine("The Loopback address is: {0}",
        test2.ToString());
else
    Console.WriteLine("Error obtaining the loopback
address");
    Console.WriteLine("The Local IP address is: {0}\n",
        myself.ToString());
    if (myself == test2)
        Console.WriteLine("The loopback address is the same as
local address.\n");
    else
        Console.WriteLine("The loopback address is not the local
address.\n");
    Console.WriteLine("The test address is: {0}",
        test1.ToString());
    Console.WriteLine("Broadcast address: {0}",
        test3.ToString());
    Console.WriteLine("The ANY address is: {0}",
        test4.ToString());
    Console.WriteLine("The NONE address is: {0}",
        test5.ToString());
}}
```

```
Microsoft Visual Studio Debug Console
The Loopback address is: 127.0.0.1
The Local IP address is: fe80::3c0c:effd:153e:6c52%14

The loopback address is not the local address.

The test address is: 192.168.1.1
Broadcast address: 255.255.255.255
The ANY address is: 0.0.0.0
The NONE address is: 255.255.255.255

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 2380) exited with
code 0.
Press any key to close this window . . .
```

OUTPUT

IPENDPOINT

- Two constructors are used to create IPEndPoint instances:
 - IPEndPoint(long address, int port)
 - IPEndPoint(IPAddress address, int port)
- Both constructors use two parameters: an IP address value, represented as either a long value or an IPAddress object;
- and the integer port number.

```
using System;
using System.Net;
class IPEndPointSample {
    public static void Main() {
        IPAddress test1 = IPAddress.Parse("192.168.1.1");
        IPEndPoint ie = new IPEndPoint(test1, 8000);
        Console.WriteLine("The IPEndPoint is: {0}", ie.ToString());
        Console.WriteLine("The AddressFamily is: {0}",
ie.AddressFamily);
        Console.WriteLine("The address is: {0}, and the port is:
{1}\n", ie.Address, ie.Port);
        Console.WriteLine("The min port number is: {0}",
IPEndPoint.MinPort);
        Console.WriteLine("The max port number is: {0}\n",
IPEndPoint.MaxPort);
        ie.Port = 80;
        Console.WriteLine("The changed IPEndPoint value is: {0} ",
ie.ToString());
        SocketAddress sa = ie.Serialize();
        Console.WriteLine("The SocketAddress is: {0}",
sa.ToString());    } }
```

```
Microsoft Visual Studio Debug Console

The IPEndPoint is: 192.168.1.1:8000
The AddressFamily is: InterNetwork
The address is: 192.168.1.1, and the port is: 8000

The min port number is: 0
The max port number is: 65535

The changed IPEndPoint value is: 192.168.1.1:80
The SocketAddress is: InterNetwork:16:{0,80,192,168,1,1,0,0,0,0,0,0,0,0}

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 2952) exited with
code 0.
Press any key to close this window . . .
```

OUTPUT

**LET'S USE
THE
SOCKETS**



USING C# SOCKETS

The Socket class constructor is as follows:

`Socket(AddressFamily af, SocketType st, ProtocolType pt)`

It uses three parameters to define the type of socket to create:

- An *AddressFamily* to define the network type
- A *SocketType* to define the type of data connection
- A *ProtocolType* to define a specific network protocol

SocketType	Protocoltype	Description
Dgram	Udp	Connectionless communication
Stream	Tcp	Connection-oriented communication
Raw	Icmp	Internet Control Message Protocol
Raw	Raw	Plain IP packet communication

IP SOCKET DEFINITION COMBINATIONS

SOCKET PROPS.

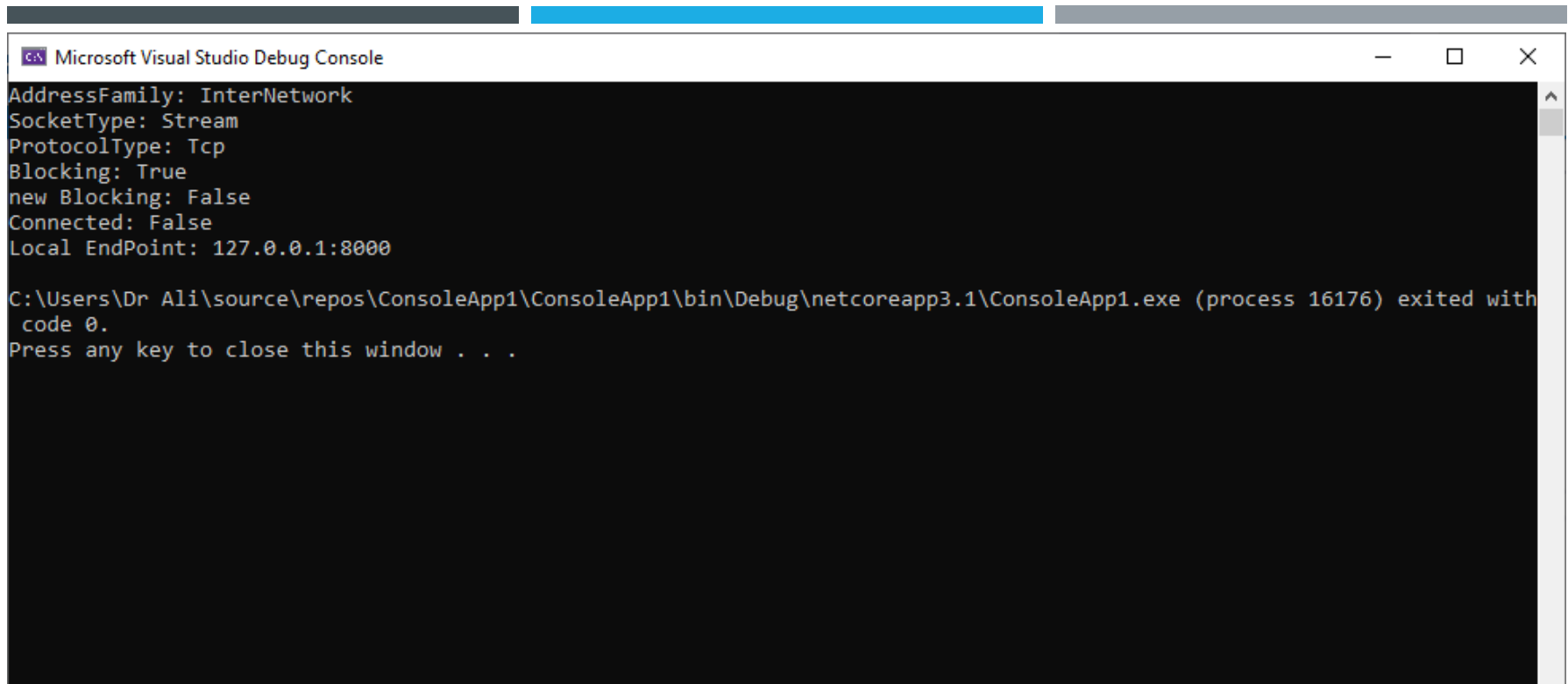
```
Socket newsock = Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

Socket Properties

Property	Description
AddressFamily	Gets the address family of the Socket
Available	Gets the amount of data that is ready to be read
Blocking	Gets or sets whether the Socket is in blocking mode
Connected	Gets a value that indicates if the Socket is connected to a remote device
Handle	Gets the operating system handle for the Socket
LocalEndPoint	Gets the local EndPoint object for the Socket
ProtocolType	Gets the protocol type of the Socket
RemoteEndPoint	Gets the remote EndPoint information for the Socket
SocketType	Gets the type of the Socket



CAN'T WAIT LET'S CODE !



Microsoft Visual Studio Debug Console

```
AddressFamily: InterNetwork
SocketType: Stream
ProtocolType: Tcp
Blocking: True
new Blocking: False
Connected: False
Local EndPoint: 127.0.0.1:8000

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 16176) exited with
code 0.
Press any key to close this window . . .
```

OUTPUT

C# SOCKET EXCEPTIONS

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SocketExcept
{
    public static void Main()
    {
        IPAddress host = IPAddress.Parse("195.168.1.1");
        IPEndPoint hostep = new IPEndPoint(host, 8000);
        Socket sock = new
Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
        try{
            sock.Connect(hostep);
        }
        catch (SocketException e){
            Console.WriteLine("Problem connecting to host");
            Console.WriteLine(e.ToString());
            sock.Close();
            return;
        }
    }
}
```

```
try{
    sock.Send(Encoding.ASCII.GetBytes("testing"));
}
catch (SocketException e){
    Console.WriteLine("Problem sending data");
    Console.WriteLine(e.ToString());
    sock.Close();
    return;
}
sock.Close();
}
```

```
Microsoft Visual Studio Debug Console
Problem connecting to host
System.Net.Internals.SocketExceptionFactory+ExtendedSocketException (10061): No connection could be made because the target machine actively refused it. 195.168.1.1:8000
    at System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress)
    at System.Net.Sockets.Socket.Connect(EndPoint remoteEP)
    at SocketExcept.Main() in C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\Program.cs:line 13

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 15608) exited with code 0.
Press any key to close this window . . .
```

OUTPUT

ECHO SERVER

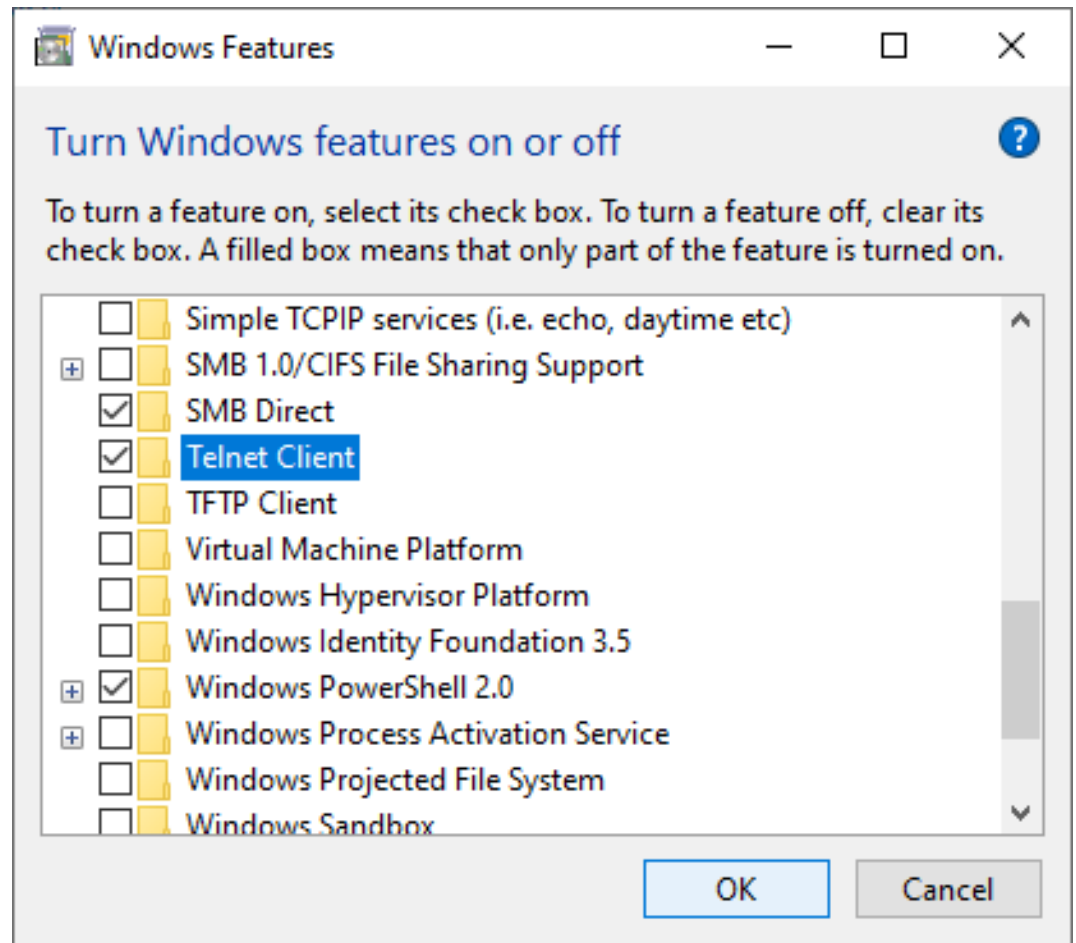
```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleTcpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Waiting for a client...");
        Socket client = newsock.Accept();
        IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
```



```
string welcome = "Welcome to my test server";
data = Encoding.ASCII.GetBytes(welcome);
client.Send(data, data.Length, SocketFlags.None);
while (true)
{
    data = new byte[1024];
    recv = client.Receive(data);
    if (recv == 0)
        break;
    Console.WriteLine(Encoding.ASCII.GetString(data, 0,
recv));
    client.Send(data, recv, SocketFlags.None);
}
Console.WriteLine("Disconnected from {0}", clientep.Address);
client.Close();
newsock.Close();
}
}
```

ENABLE TELNET CLIENT

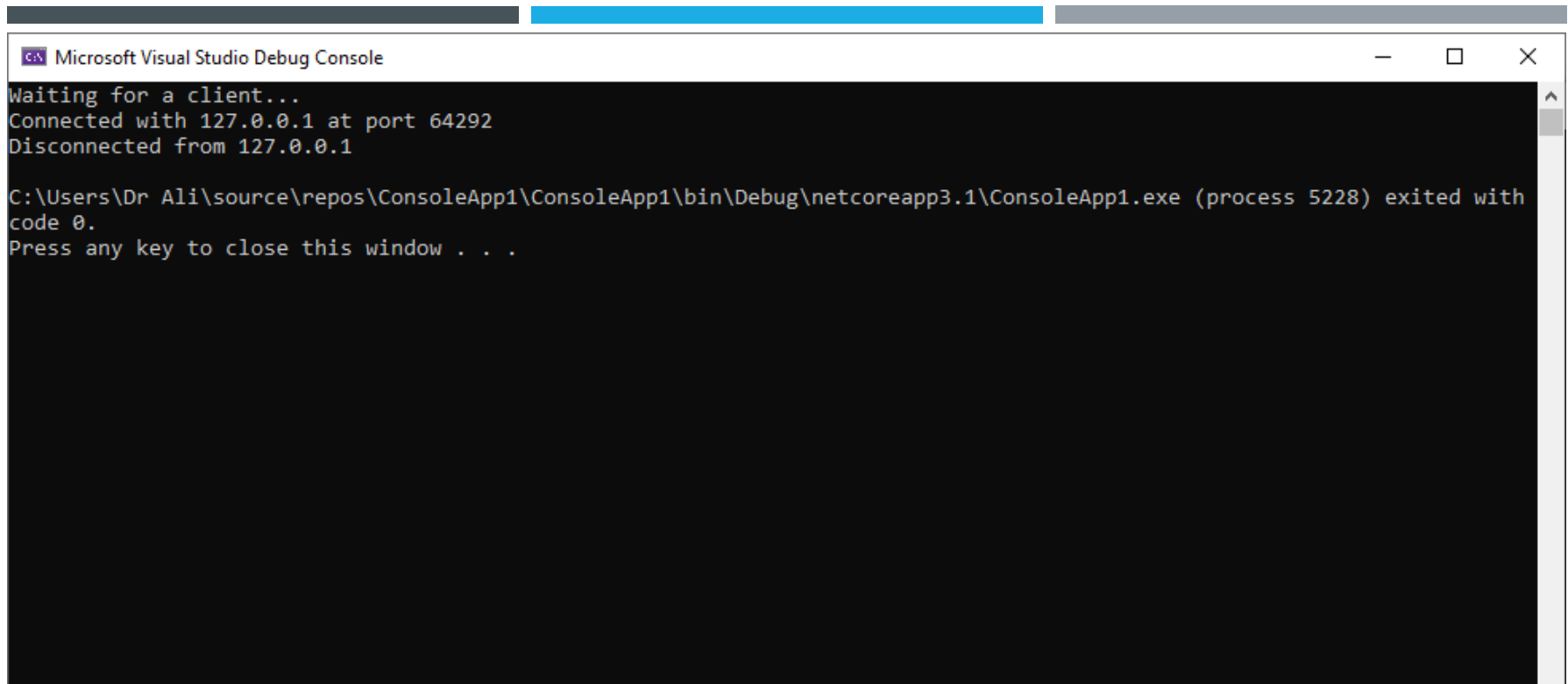
- How to Install Telnet in Windows 10
- Open “Control Panel”.
- Open “Programs”.
- Select the “Turn Windows features on or off ” option.
- Check the “Telnet Client” box.
- Click “OK”. A box will appear that says “Windows features” and “Searching for required files“. When complete, the Telnet client should be installed in Windows.





TELNET UR SERVER

- Open the CMD
- C:\>telnet 127.0.0.1 9050



The image shows a screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the Visual Studio logo and the text "Microsoft Visual Studio Debug Console". The console output is as follows:

```
Waiting for a client...
Connected with 127.0.0.1 at port 64292
Disconnected from 127.0.0.1

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 5228) exited with
code 0.
Press any key to close this window . . .
```

OUTPUT



THANK U

NEXT TOPICS

- TCP CLIENT
- When TCP Goes Bad
- Using Fixed-Sized Messages

NETWORK PROGRAMMING

IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

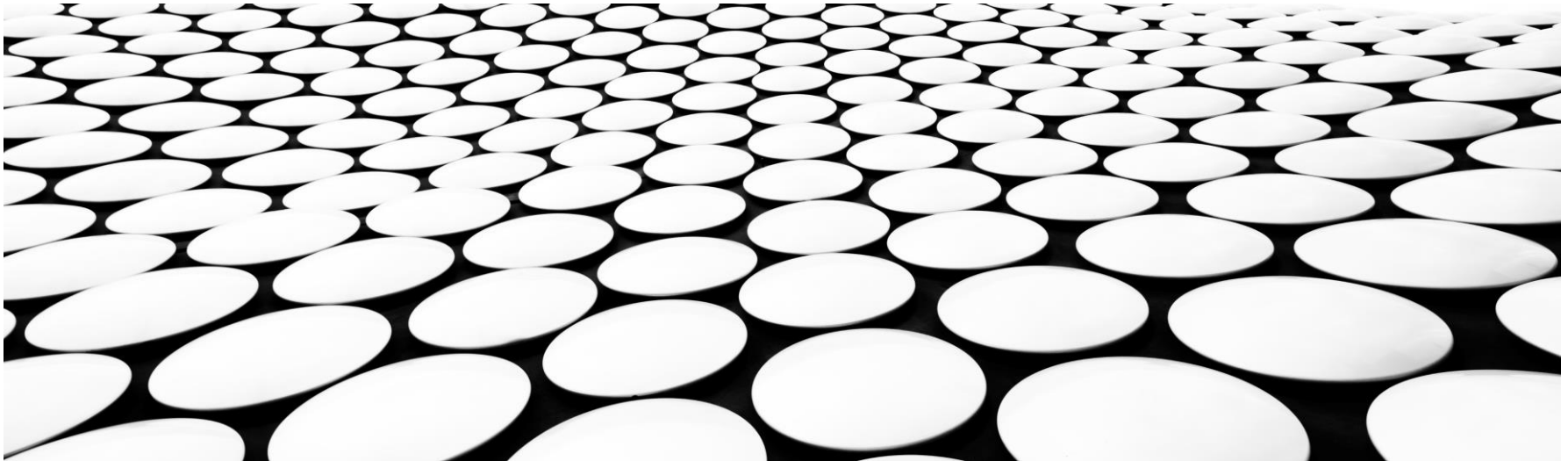
2022



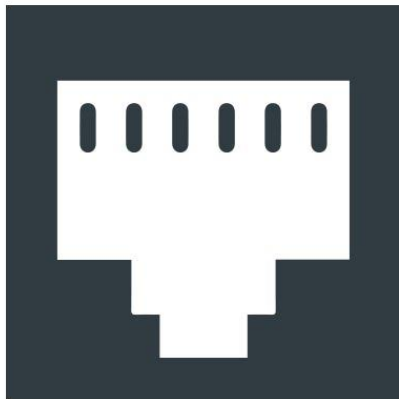
كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

LECTURE 4

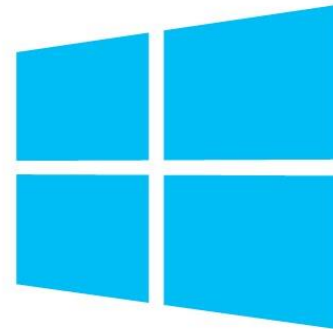
CREATING THE CLIENT . . .



SOCKETS



SOCKETS

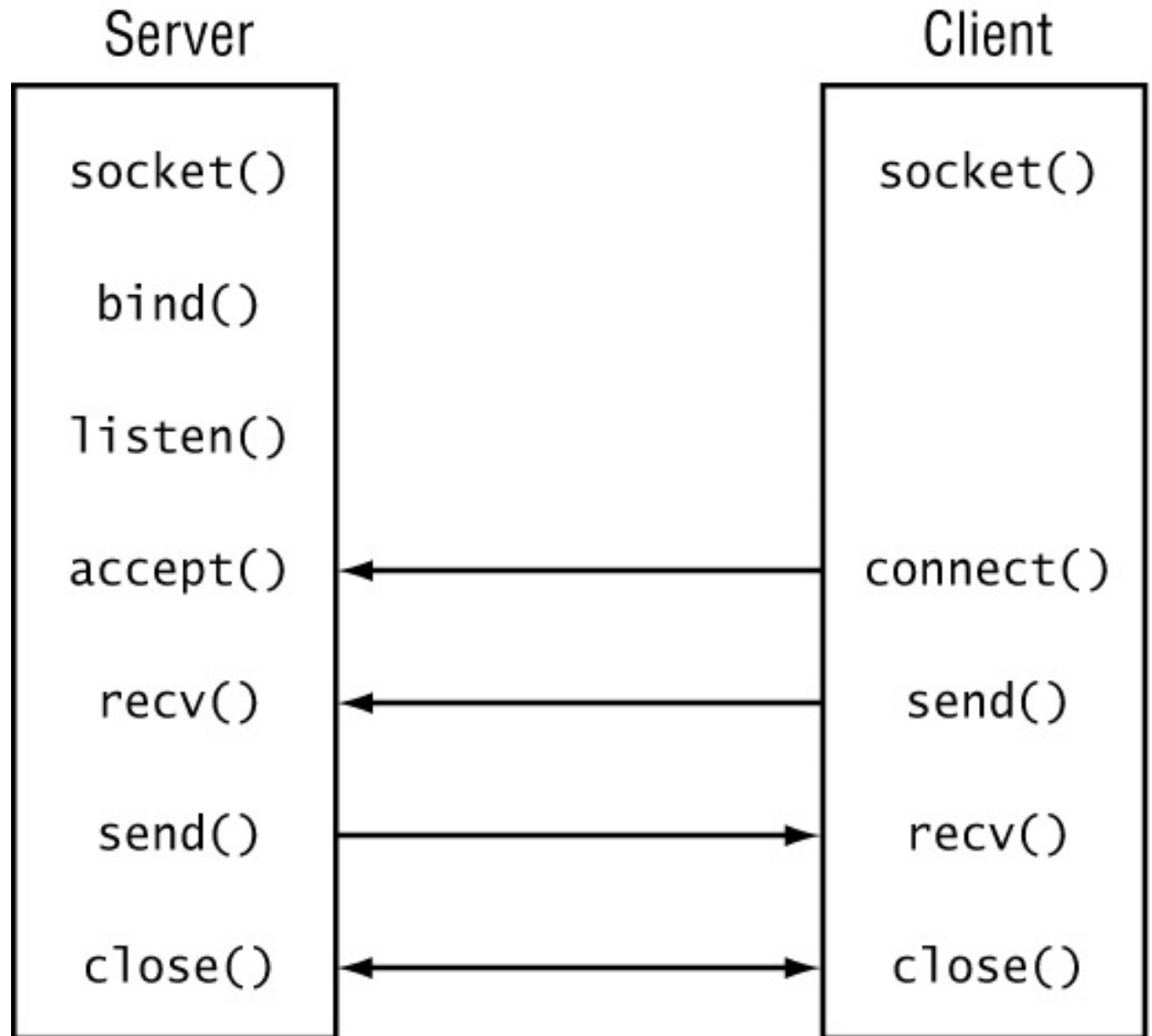


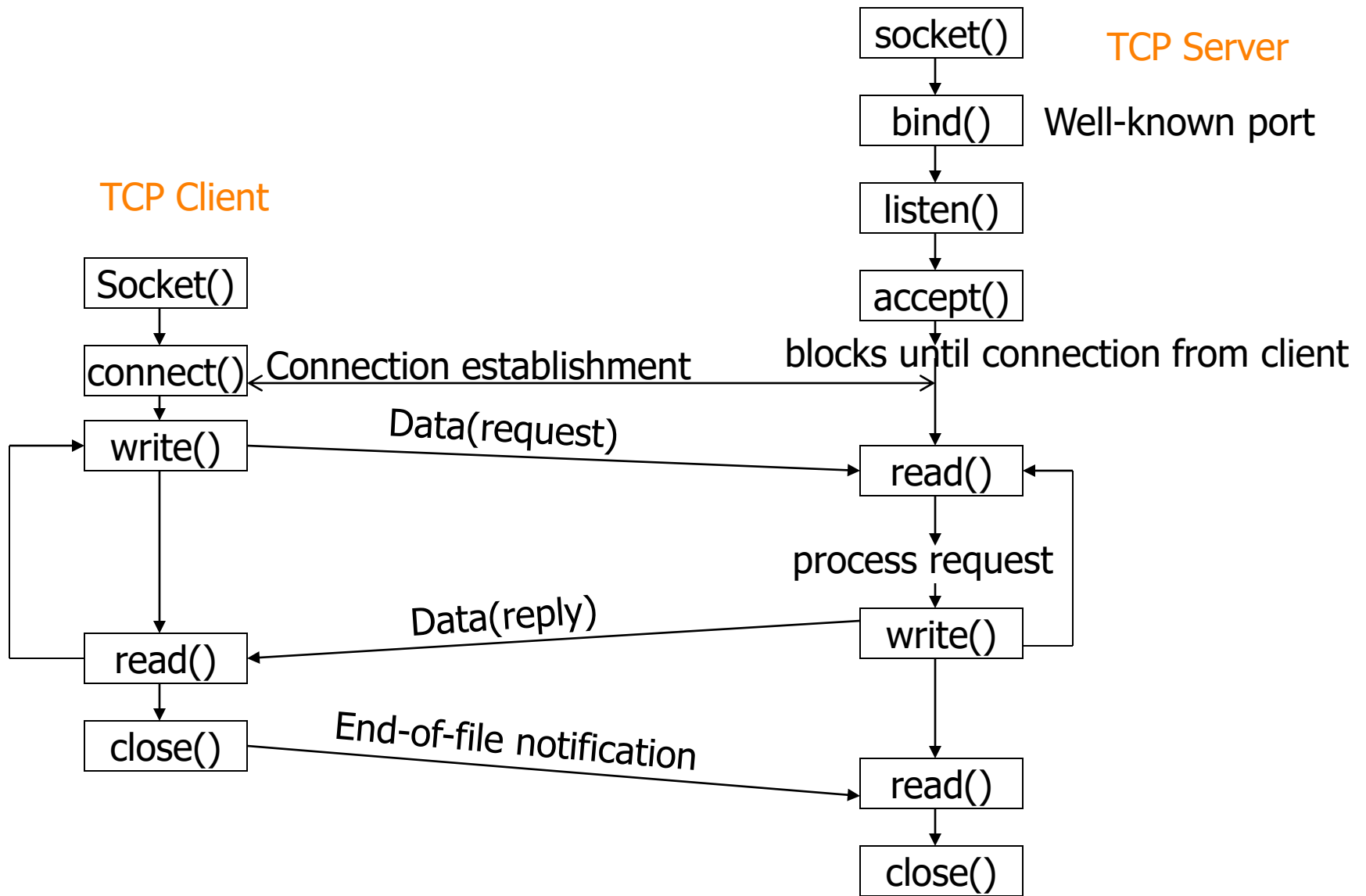


SOCKET TYPES

- **Connection-Oriented Sockets**
- **Connectionless Sockets**

STEPS

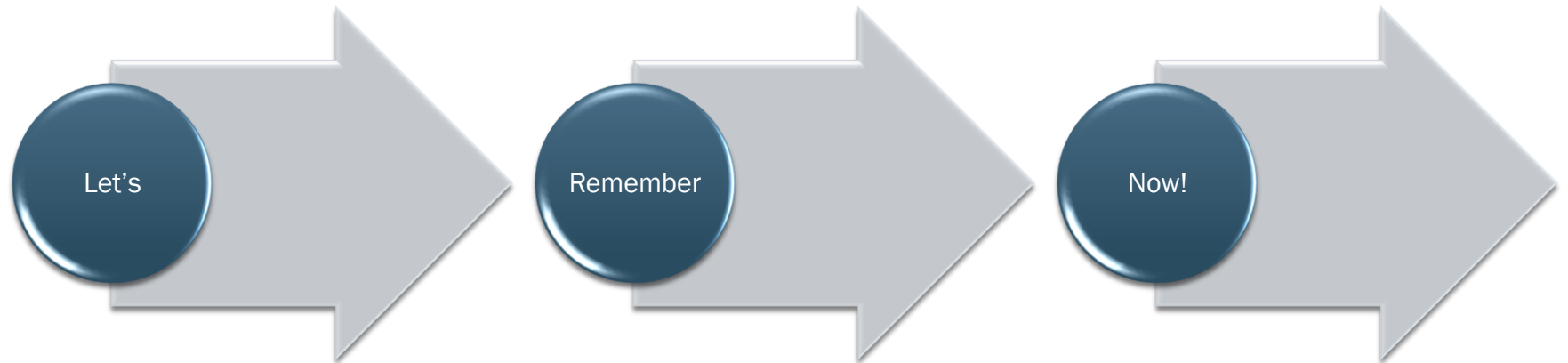






THE CLIENT FUNCTIONS

- In a connection-oriented socket, the client must bind to the specific host address and port for the application.
- For client programs, the `connect()` function is used instead of the `listen()` function:
- Once the `connect()` function succeeds, the client is connected to the server and can use the standard `send()` and `receive()` functions to transmit data back and forth with the server.



ECHO SERVER

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleTcpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Waiting for a client...");
        Socket client = newsock.Accept();
        IPEndPoint clientep = (IPEndPoint)client.RemoteEndPoint;
```

```
string welcome = "Welcome to my test server";
data = Encoding.ASCII.GetBytes(welcome);
client.Send(data, data.Length, SocketFlags.None);
while (true)
{
    data = new byte[1024];
    recv = client.Receive(data);
    if (recv == 0)
        break;
    Console.WriteLine(Encoding.ASCII.GetString(data, 0,
        recv));
    client.Send(data, recv, SocketFlags.None);
}
Console.WriteLine("Disconnected from {0}", clientep.Address);
client.Close();
newsock.Close();
}
}
```


NEXT TOPICS

- TCP CLIENT
- When TCP Goes Bad
- Using Fixed-Sized Messages

SIMPLE CLIENT

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
namespace client {
    class Program {
        static void Main(string[] args) {
            byte[] bytes = new byte[1024];
            IPAddress host = IPAddress.Parse("127.0.0.1");
            IPEndPoint hostep = new IPEndPoint(host, 9050);
            Socket sock = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
            sock.Connect(hostep);
            Console.WriteLine("Socket connected to {0}",
                sock.RemoteEndPoint.ToString());
            // Encode the data string into a byte array.
```

SIMPLE CLIENT CONT.

```
byte[] msg = Encoding.ASCII.GetBytes(Console.ReadLine());

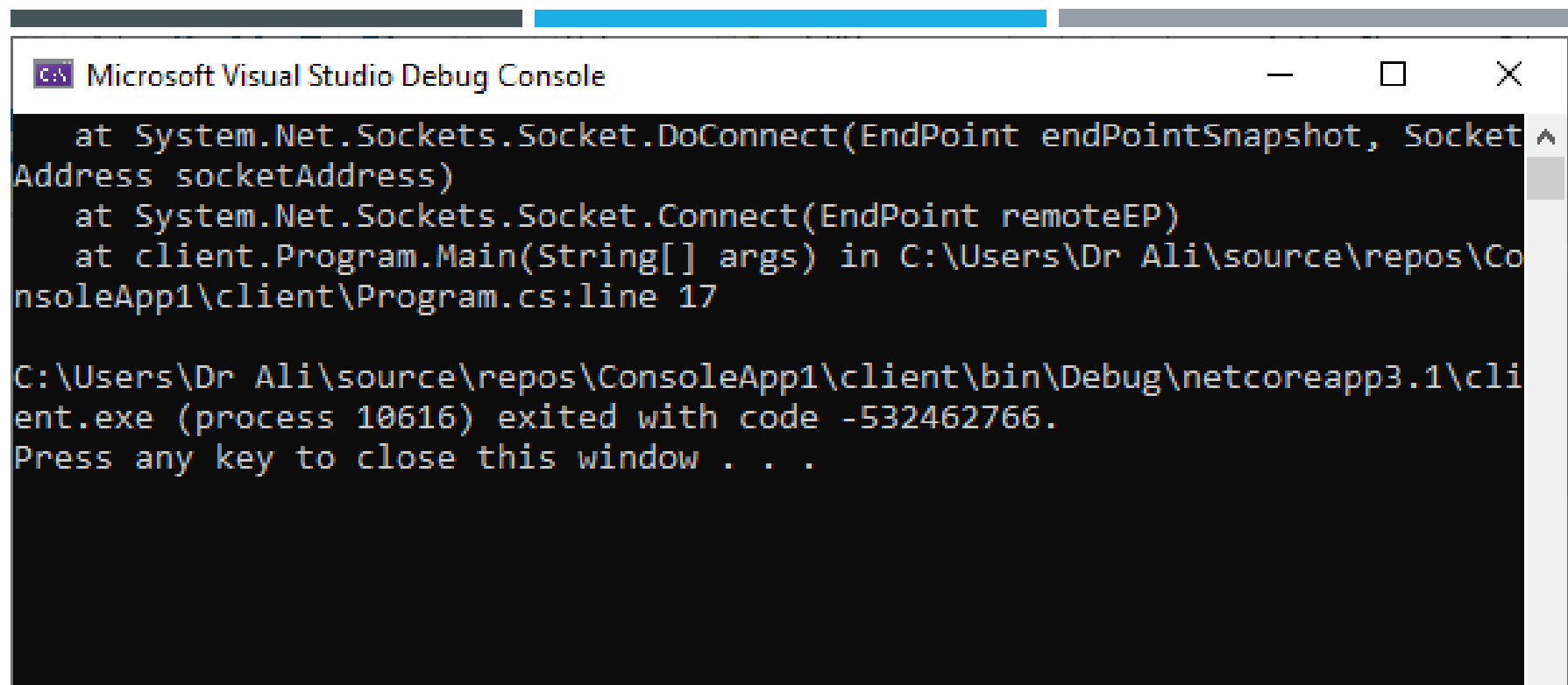
    // Send the data through the socket.
    int bytesSent = sock.Send(msg);

    // Receive the response from the remote device.
    int bytesRec = sock.Receive(bytes);
    Console.WriteLine("Echoed test =
{0}", Encoding.ASCII.GetString(bytes, 0, bytesRec));
    sock.Close();

}

}

}
```



The image shows a screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the text "Microsoft Visual Studio Debug Console" and standard window controls (minimize, maximize, close). The console output is as follows:

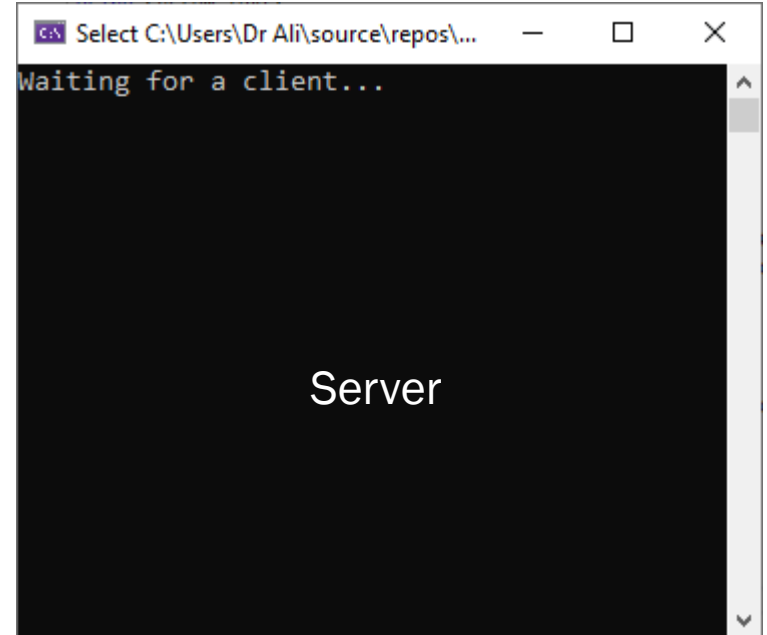
```
at System.Net.Sockets.Socket.DoConnect(EndPoint endPointSnapshot, SocketAddress socketAddress)
at System.Net.Sockets.Socket.Connect(EndPoint remoteEP)
at client.Program.Main(String[] args) in C:\Users\Dr Ali\source\repos\ConsoleApp1\client\Program.cs:line 17

C:\Users\Dr Ali\source\repos\ConsoleApp1\client\bin\Debug\netcoreapp3.1\client.exe (process 10616) exited with code -532462766.
Press any key to close this window . . .
```

RUNNING THE CLIENT FIRST, WHY?

YOU HAVE TO ..

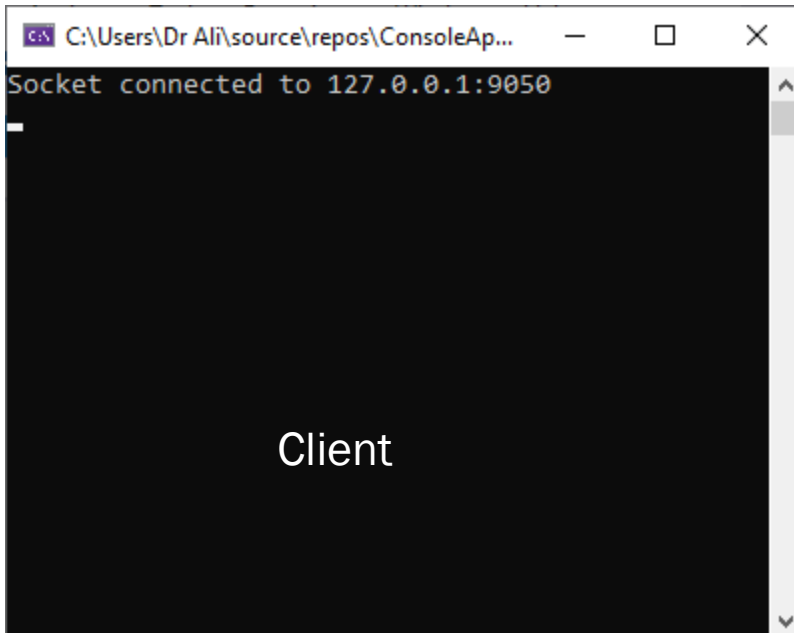
- Run the server program, it must be always on.
- Run the client ..



A screenshot of a Windows console window. The title bar shows the file path "Select C:\Users\Dr Ali\source\repos\...". The console output displays the text "Waiting for a client..." in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
Select C:\Users\Dr Ali\source\repos\...  
Waiting for a client...
```

Server



A screenshot of a Windows console window. The title bar shows the file path "C:\Users\Dr Ali\source\repos\ConsoleAp...". The console output displays the text "Socket connected to 127.0.0.1:9050" in a monospaced font. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\Users\Dr Ali\source\repos\ConsoleAp...  
Socket connected to 127.0.0.1:9050
```

Client

THE C# NETWORK STREAMS

- The .NET Framework supplies some extra classes to help out.
- This slides describes the NetworkStream class, which provides a stream interface for sockets, as well as two additional stream classes, StreamReader and StreamWriter, that can be used to send and receive text messages using TCP.

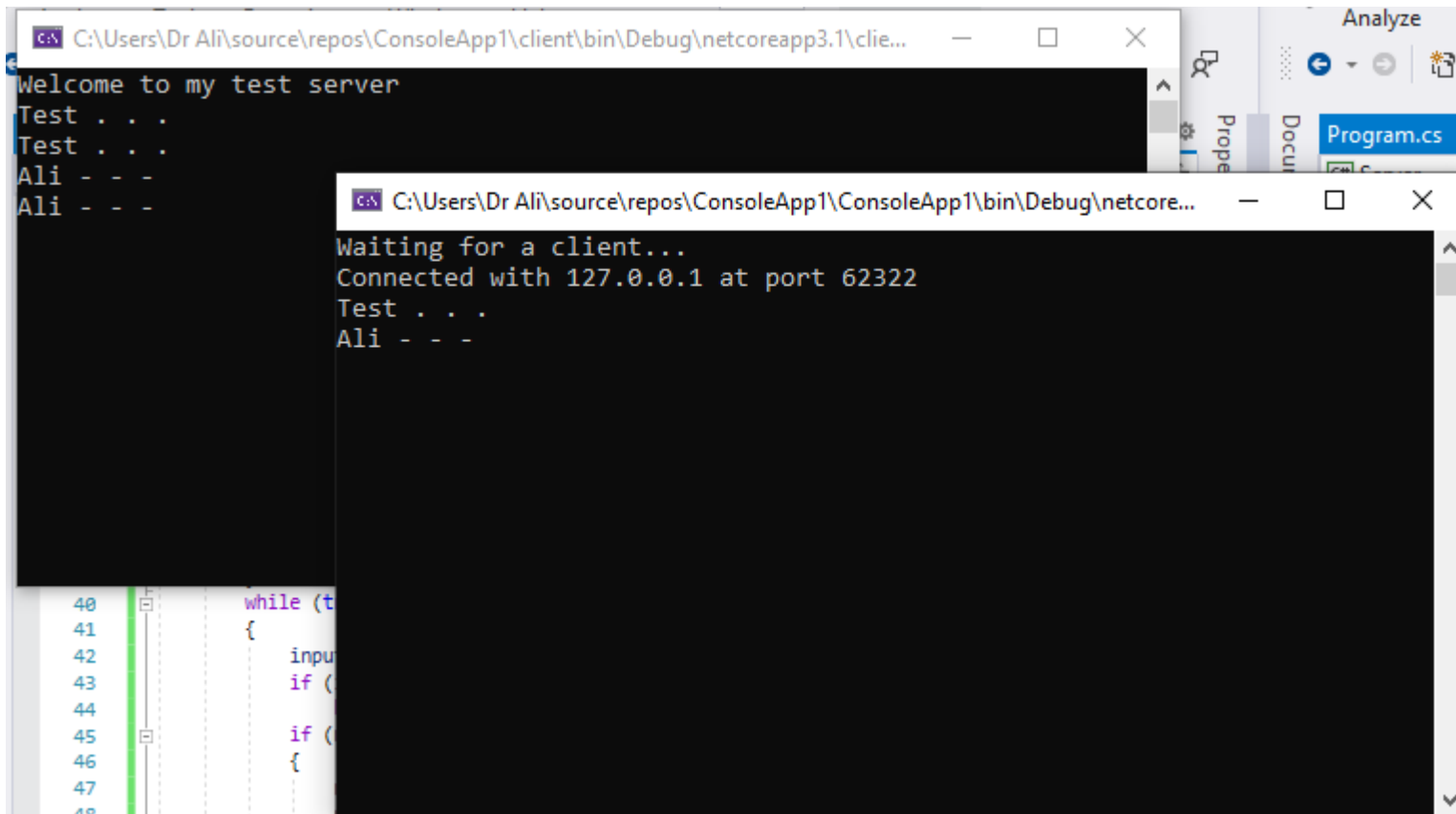
```
Socket newsock = new Socket(AddressFamily.InterNetwork,  
    SocketType.Stream, ProtocolType.Tcp);  
NetworkStream ns = new NetworkStream(newsock);
```

USING NETWORKSTREAM IN OUR CLIENT

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class NetworkStreamTcpClient {
    public static void Main() {
        byte[] data = new byte[1024];
        string input, stringData;
        int recv;
        IPEndPoint ipep = new IPEndPoint(
            IPAddress.Parse("127.0.0.1"), 9050);
        Socket server = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
        try {
            server.Connect(ipep);
        }
        catch (SocketException e) {
            Console.WriteLine("Unable to connect to server.");
            Console.WriteLine(e.ToString());
            return;
        }
        NetworkStream ns = new NetworkStream(server);
```

```
if (ns.CanRead){
    recv = ns.Read(data, 0, data.Length);
    stringData = Encoding.ASCII.GetString(data, 0, recv);
    Console.WriteLine(stringData);
}
else {
    Console.WriteLine("Error: Can't read from this socket");
    ns.Close();
    server.Close();
    return;
}
while (true) {
    input = Console.ReadLine();
    if (input == "exit")
        break;
    if (ns.CanWrite) {
        ns.Write(Encoding.ASCII.GetBytes(input), 0, input.Length);
        ns.Flush();
    }
    recv = ns.Read(data, 0, data.Length);
    stringData = Encoding.ASCII.GetString(data, 0, recv);
    Console.WriteLine(stringData);
}
Console.WriteLine("Disconnecting from server...");
ns.Close();
server.Shutdown(SocketShutdown.Both);
server.Close();    }}
```


LET'S RUN THE ENHANCED CLIENT



The screenshot displays two console windows from a Windows IDE. The background window, titled 'C:\Users\Dr Ali\source\repos\ConsoleApp1\client\bin\Debug\netcoreapp3.1\cli...', shows the client's execution. The foreground window, titled 'C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcore...', shows the server's execution. Below the windows, a portion of the source code is visible, showing a 'while' loop.

```
C:\Users\Dr Ali\source\repos\ConsoleApp1\client\bin\Debug\netcoreapp3.1\cli...
Welcome to my test server
Test . . .
Test . . .
Ali - - -
Ali - - -

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcore...
Waiting for a client...
Connected with 127.0.0.1 at port 62322
Test . . .
Ali - - -

40 while (t
41 {
42     input
43     if (
44
45     if (
46     {
47
48
```



THE STREAMREADER AND STREAMWRITER CLASSES

- The two helper classes can be used with any stream, say our network stream.
- The next few slides rewrites the server program to be more efficient.

STREAM SERVER

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpSrvr {
    public static void Main(){
        string data;
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
        newsock.Bind(ipep);
        newsock.Listen(10);
        Console.WriteLine("Waiting for a client...");
        Socket client = newsock.Accept();
        IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;
        Console.WriteLine("Connected with {0} at port {1}",
            newclient.Address, newclient.Port);
        NetworkStream ns = new NetworkStream(client);
```

```
StreamReader sr = new StreamReader(ns);
StreamWriter sw = new StreamWriter(ns);
string welcome = "Welcome to my test server";
sw.WriteLine(welcome);
sw.Flush();
while (true)
{
    try { data = sr.ReadLine(); }
    catch (IOException) { break; }
    Console.WriteLine(data);
    sw.WriteLine(data);
    sw.Flush();
}
Console.WriteLine("Disconnected from {0}",
newclient.Address);
sw.Close();
sr.Close();
ns.Close();
}
}
```

STREAM CLIENT

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
class StreamTcpClient{
    public static void Main()    {
        string data;
        string input;
        IPEndPoint ipep = new IPEndPoint(
            IPAddress.Parse("127.0.0.1"), 9050);
        Socket server = new Socket(AddressFamily.InterNetwork,
            SocketType.Stream, ProtocolType.Tcp);
        try    {
            server.Connect(ipep);
        }
        catch (SocketException e)
        {
            Console.WriteLine("Unable to connect to server.");
            Console.WriteLine(e.ToString());
            return;
        }
    }
}
```

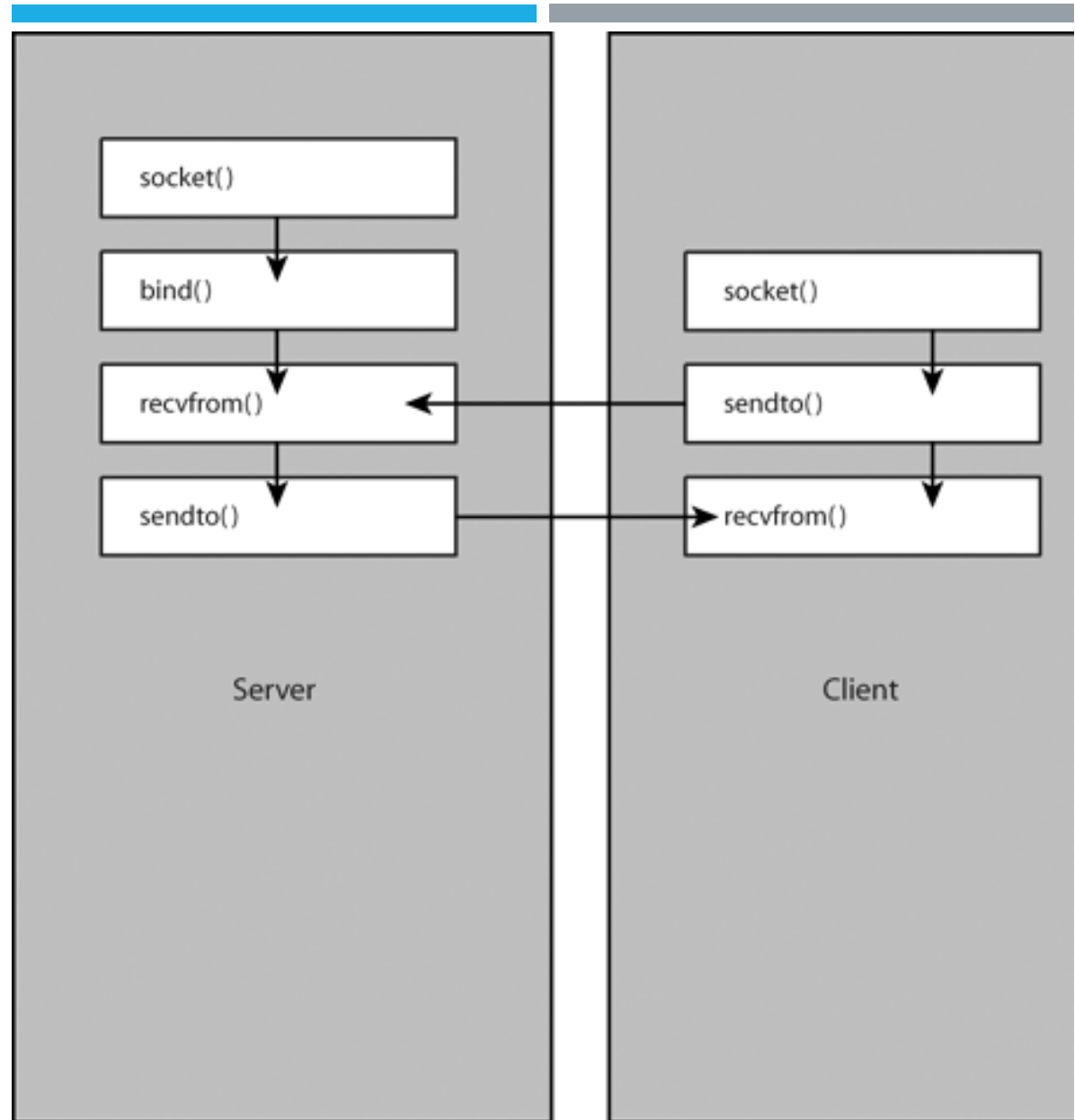
```
NetworkStream ns = new NetworkStream(server);
    StreamReader sr = new StreamReader(ns);
    StreamWriter sw = new StreamWriter(ns);
    data = sr.ReadLine();
    Console.WriteLine(data);
    while (true)
    {
        input = Console.ReadLine();
        if (input == "exit")
            break;
        sw.WriteLine(input);
        sw.Flush();
        data = sr.ReadLine();
        Console.WriteLine(data);
    }
    Console.WriteLine("Disconnecting from server...");
    sr.Close();
    sw.Close();
    ns.Close();
    server.Shutdown(SocketShutdown.Both);
    server.Close();
}}
```



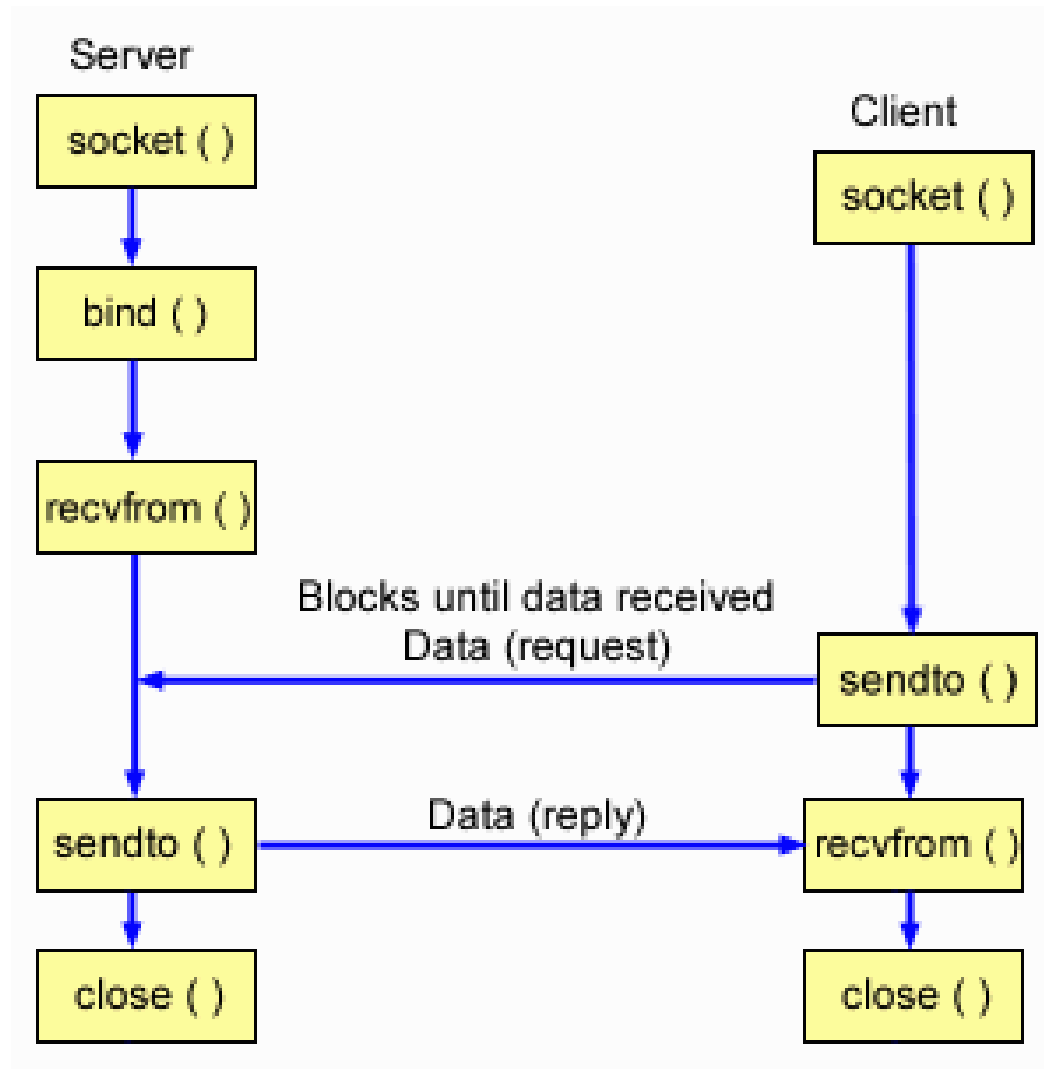
CONNECTIONLESS SOCKETS

- Connectionless sockets allow the sending of messages in self-contained packets.
- A single read method reads the entire message sent by a single sent method.
- This helps you avoid the hassle of trying to match message boundaries in packets.
- Unfortunately, UDP packets are not guaranteed to arrive at their destination.
- Many factors, such as busy networks, can prevent the packet from making it to its destination.

HOW IT RUNS



ITS NOT SERVER, JUST WHO ARE THE ONE ISSUING THE BIND AND THE FIRST RECEIVE



UDP SERVER

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpSrvr
{
    public static void Main()
    {
        int recv;
        byte[] data = new byte[1024];
        IPEndPoint ipep = new IPEndPoint(IPAddress.Any, 9050);
        Socket newsock = new Socket(AddressFamily.InterNetwork,
                                     SocketType.Dgram, ProtocolType.Udp);
        newsock.Bind(ipep);
        Console.WriteLine("Waiting for a client...");
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint Remote = (EndPoint)(sender);
        recv = newsock.ReceiveFrom(data, ref Remote);
    }
}
```

```
Console.WriteLine("Message received from {0}:",
    Remote.ToString());
    Console.WriteLine(Encoding.ASCII.GetString(data, 0,
recv));
    string welcome = "Welcome to my test server";
    data = Encoding.ASCII.GetBytes(welcome);
    newsock.SendTo(data, data.Length, SocketFlags.None,
Remote);
    while (true)
    {
        data = new byte[1024];
        recv = newsock.ReceiveFrom(data, ref Remote);

        Console.WriteLine(Encoding.ASCII.GetString(data, 0,
recv));
        newsock.SendTo(data, recv, SocketFlags.None, Remote);
    }
}
```

UDP CLIENT

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class SimpleUdpClient
{
    public static void Main()
    {
        byte[] data = new byte[1024];
        string input, stringData;
        IPEndPoint ipep = new IPEndPoint(
            IPAddress.Parse("127.0.0.1"), 9050);
        Socket server = new Socket(AddressFamily.InterNetwork,
            SocketType.Dgram, ProtocolType.Udp);
        string welcome = "Hello, are you there?";
        data = Encoding.ASCII.GetBytes(welcome);
        server.SendTo(data, data.Length, SocketFlags.None, ipep);
        IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
        EndPoint Remote = (EndPoint)sender;
        data = new byte[1024];
    }
}
```

```
int recv = server.ReceiveFrom(data, ref Remote);
    Console.WriteLine("Message received from {0}:",
Remote.ToString());
    Console.WriteLine(Encoding.ASCII.GetString(data, 0, recv));
    while (true)
    {
        input = Console.ReadLine();
        if (input == "exit")
            break;
        server.SendTo(Encoding.ASCII.GetBytes(input), Remote);
        data = new byte[1024];
        recv = server.ReceiveFrom(data, ref Remote);
        stringData = Encoding.ASCII.GetString(data, 0, recv);
        Console.WriteLine(stringData);
    }
    Console.WriteLine("Stopping client");
    server.Close();
}
}
```

SECTION WORK THIS WEEK

- Create clean UI for the following programs:
 - Implement and run all the code snippets listed in this lecture.
 - Can you tune the last stream server and client to send a file!
 - Try to send a file using client server program
 - Your project should send the file from server to client.
 - Make use of file streams you learnt before.
 - **Compress the files and submit them on the subject team (Eng Salma will create the Dir)**
 - **You will take 5 points /100 if U did this**
 - **Work at home, deliver & submit on section (grading will be on section)**

NETWORK PROGRAMMING

IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

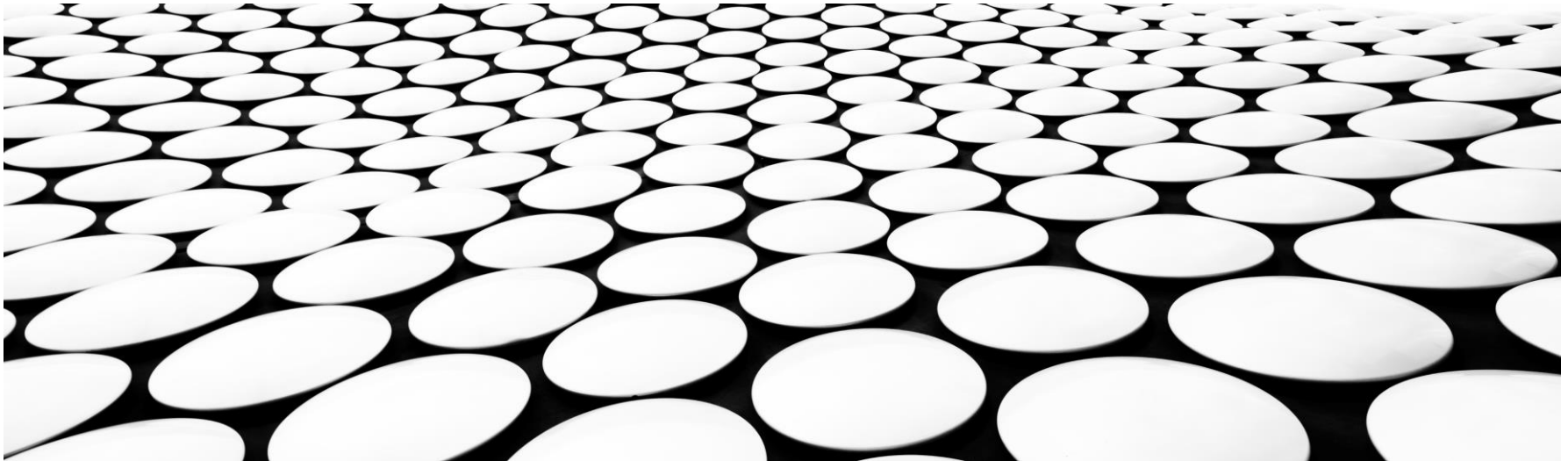
2022



كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

LECTURE 5


ASYNCHRONOUS SOCKETS





THE BLOCKING SOCKET MODE

- Sockets in blocking mode will wait forever to complete their functions, holding up other functions within the application program until they complete.
- Many programs can work quite efficiently in this mode, but for applications that work in the Windows programming environment, this can be a problem.
- Asynchronous Socket methods allow a network program to continue rather than waiting for a network operation to be performed.
- Guess which is best?

- 
- Just as events can trigger delegates, .NET also provides a way for methods to trigger delegates.
 - A delegate defines the method to be called once an event occurred.

```
checkit.Click += new EventHandler(ButtonOnClick);
```


- When the customer clicks the button, the program control moves to the ButtonOnClick() method:

```
void ButtonOnClick(object obj, EventArgs ea)
{
    results.Items.Add(data.Text);
    data.Clear();
}
```



ASYNC CALLBACK CLASS

- The .NET AsyncCallback class allows methods to start an asynchronous function and supply a delegate method to call when the asynchronous function completes.
- The Socket class utilizes the method defined in the AsyncCallback to allow network functions to operate asynchronously in background processing.
- It signals the OS when the network functions have completed and passes program control to the AsyncCallback method to finish the network function.

- 
- The Socket asynchronous methods split common network programming functions into two pieces:
 - A **Begin** method that starts the network function and registers the AsyncCallback method
 - An **End** method that completes the function when the AsyncCallback method is called

.NET ASYNCHRONOUS SOCKET METHODS

Requests Started By...	Description of Request	Requests Ended BY...
BeginAccept()	To accept an incoming connection	EndAccept()
BeginConnect()	To connect to a remote host	EndConnect()
BeginReceive()	To retrieve data from a socket	EndReceive()
BeginReceiveFrom()	To retrieve data from a specific remote host	EndReceiveFrom()
BeginSend()	To send data from a socket	EndSend()
BeginSendTo()	To send data to a specific remote host	EndSendTo()

THE BEGINACCEPT() AND ENDACCEPT() METHODS

```
Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
sock.Bind(iep);
sock.Listen(5);
sock.BeginAccept(new AsyncCallback(CallAccept), sock);

private static void CallAccept(IAsyncResult iar)
{
    Socket server = (Socket)iar.AsyncState;
    Socket client = server.EndAccept(iar);
}
```

THE BEGINCONNECT() AND ENDCONNECT() METHODS

```
Socket newsock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
newsock.BeginConnect(iep, new AsyncCallback(Connected), newsock);
```

```
public static void Connected(IAsyncResult iar)
{
    Socket sock = (Socket)iar.AsyncState;
    try
    {
        sock.EndConnect(iar);
    }
    catch (SocketException)
    {
        Console.WriteLine("Unable to connect to host");
    }
}
```

SENDING AND RECEIVING DATA

```
sock.BeginSend(data, 0, data.Length, SocketFlags.None,  
    new AsyncCallback(SendData), sock);
```

```
private static void SendData(IAsyncResult iar)  
{  
    Socket server = (Socket)iar.AsyncState;  
    int sent = server.EndSend(iar);  
}
```




OTHER ASYNCH CALLS

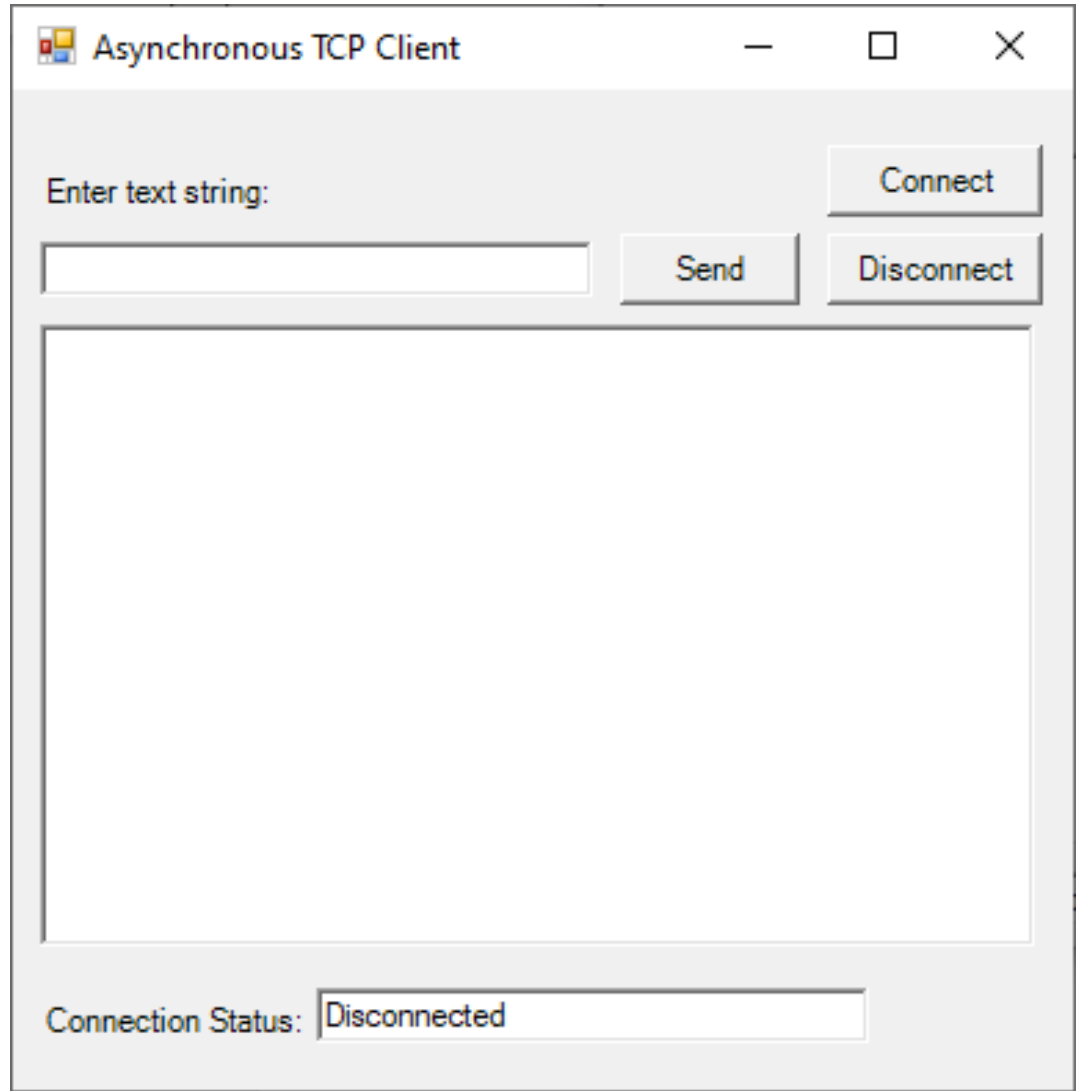
- **The BeginSendTo() and EndSendTo() Methods**
- **The BeginReceive() and EndReceive() Methods**
- **The BeginReceiveFrom() and EndReceiveFrom() Methods**



EXAMPLE

- In the next few slides, we will create a windows application for both client and server.
- We will notice that our GUI did not freeze during the blocking socket calls.

THE CLIENT PROGRAM



The screenshot shows a Windows-style application window titled "Asynchronous TCP Client". The window has a standard title bar with minimize, maximize, and close buttons. The main interface includes a label "Enter text string:" followed by a text input field. To the right of the input field are three buttons: "Connect", "Send", and "Disconnect". Below these controls is a large, empty rectangular area, likely for displaying received data or logs. At the bottom of the window, there is a label "Connection Status:" followed by a text field that currently displays the word "Disconnected".

Asynchronous TCP Client

Enter text string:

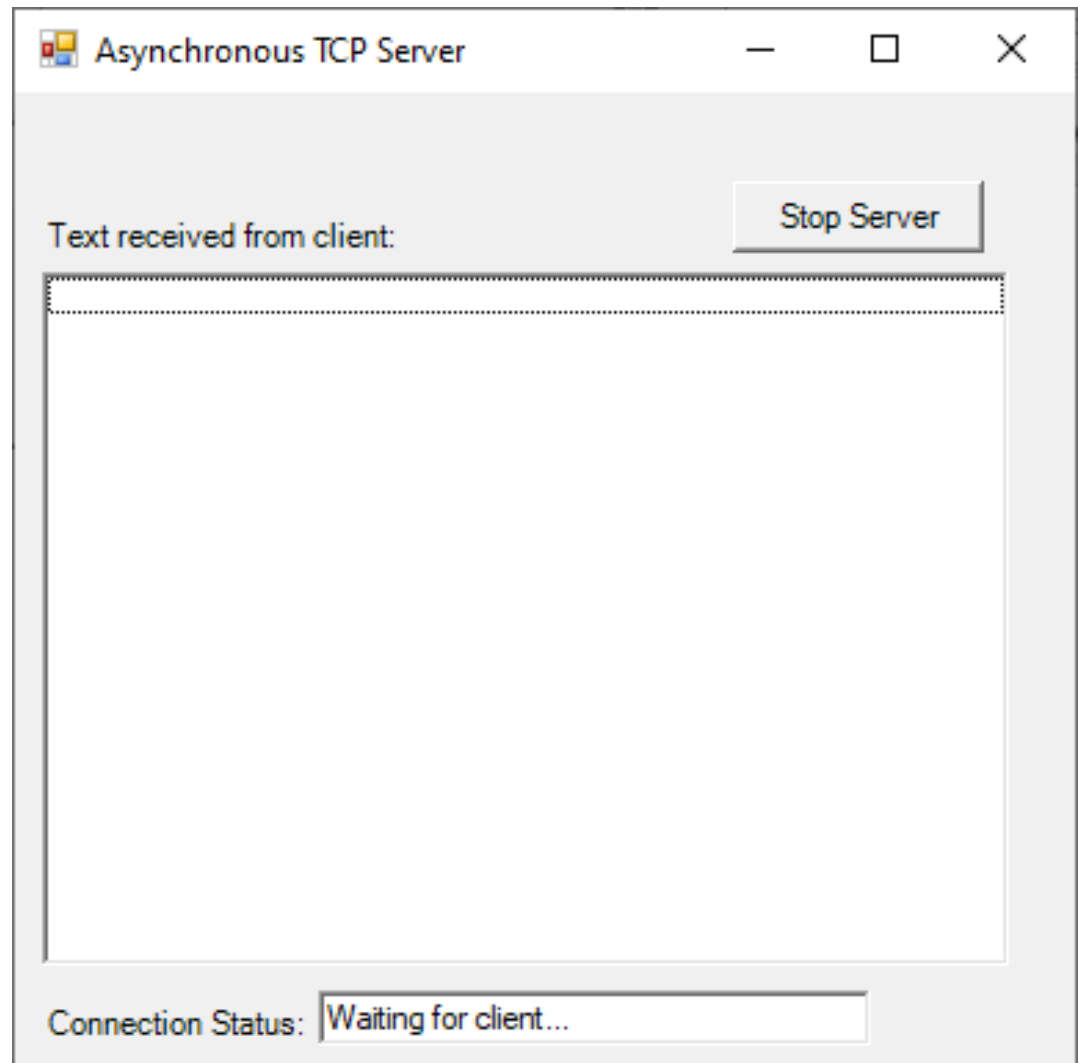
Connect

Send

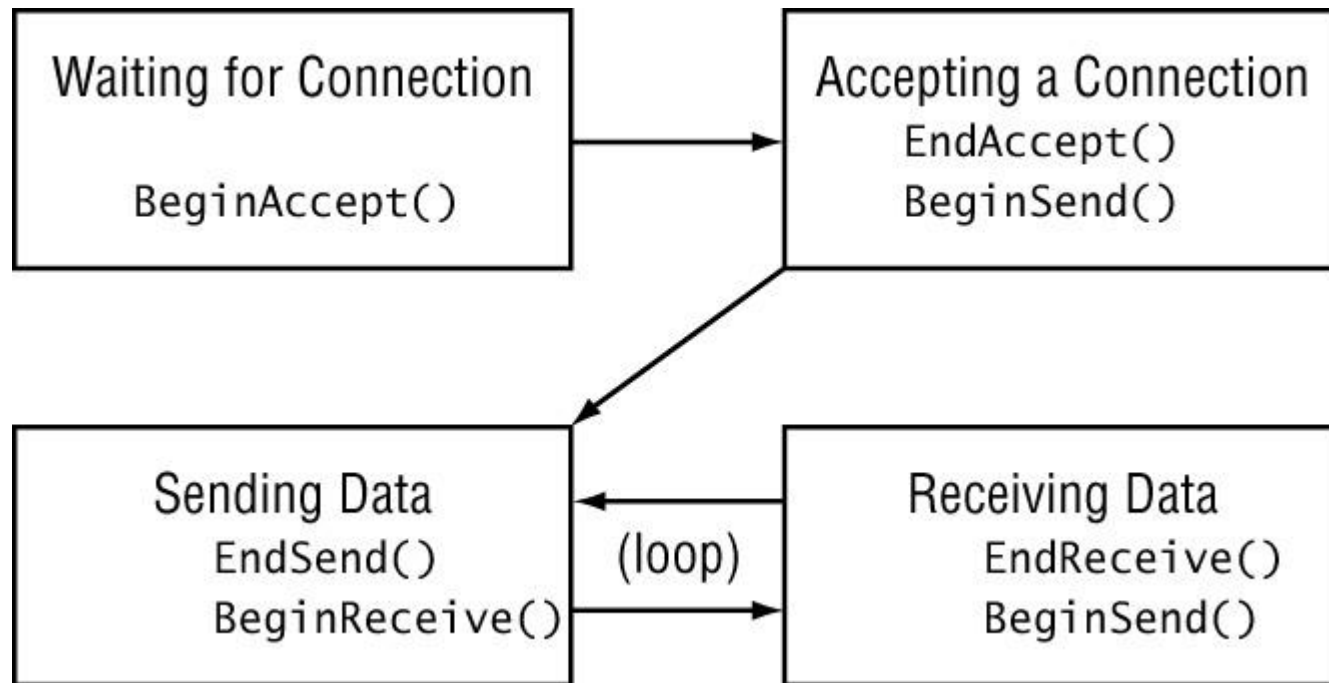
Disconnect

Connection Status: Disconnected

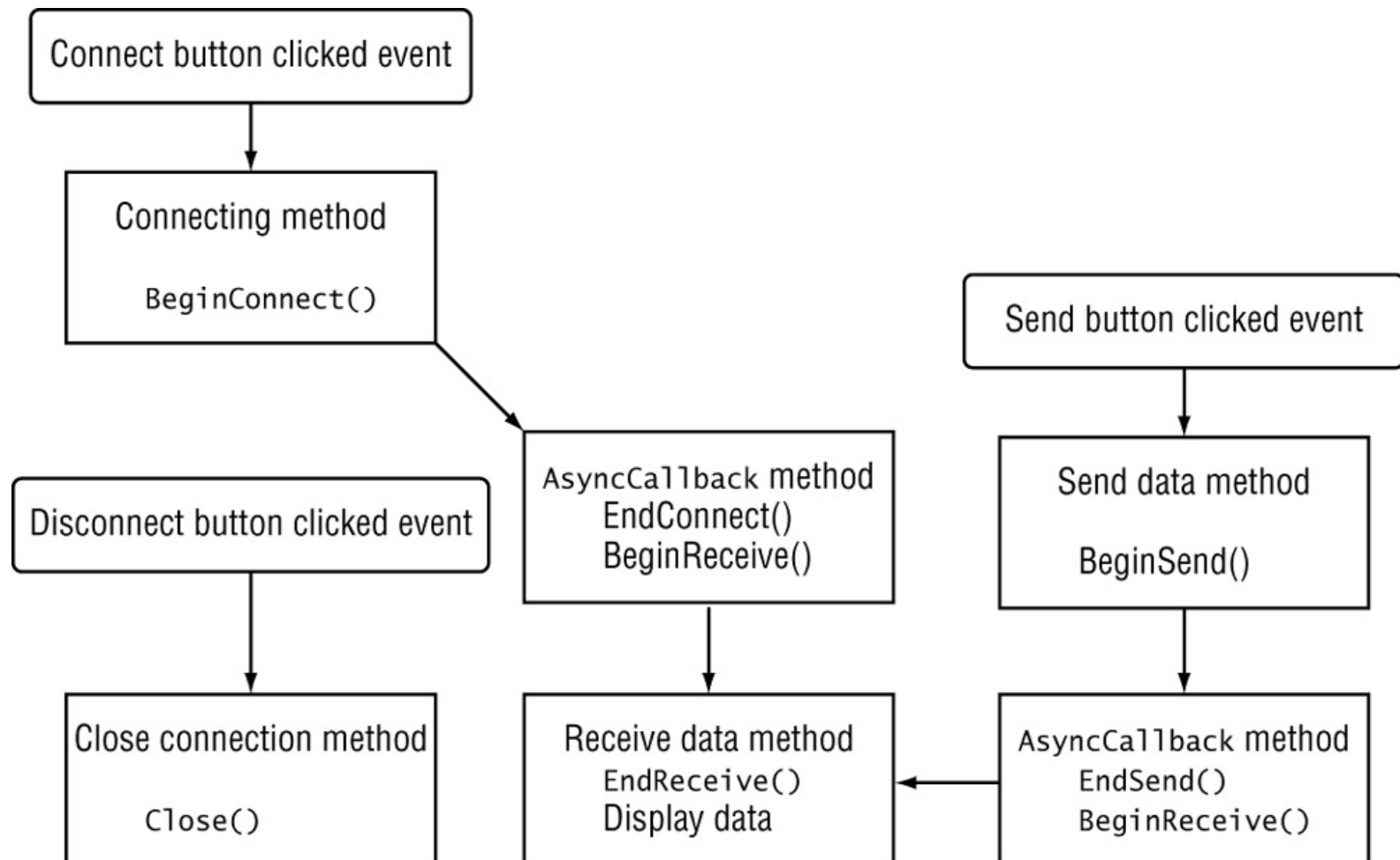
THE SERVER PROGRAM



THE SERVER LOOP



THE CLIENT





THE SOURCE CODE

- [Client.cs](#)
- [Server.cs](#)



NEXT TOPIC

- Multithreaded server

NETWORK PROGRAMMING

IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

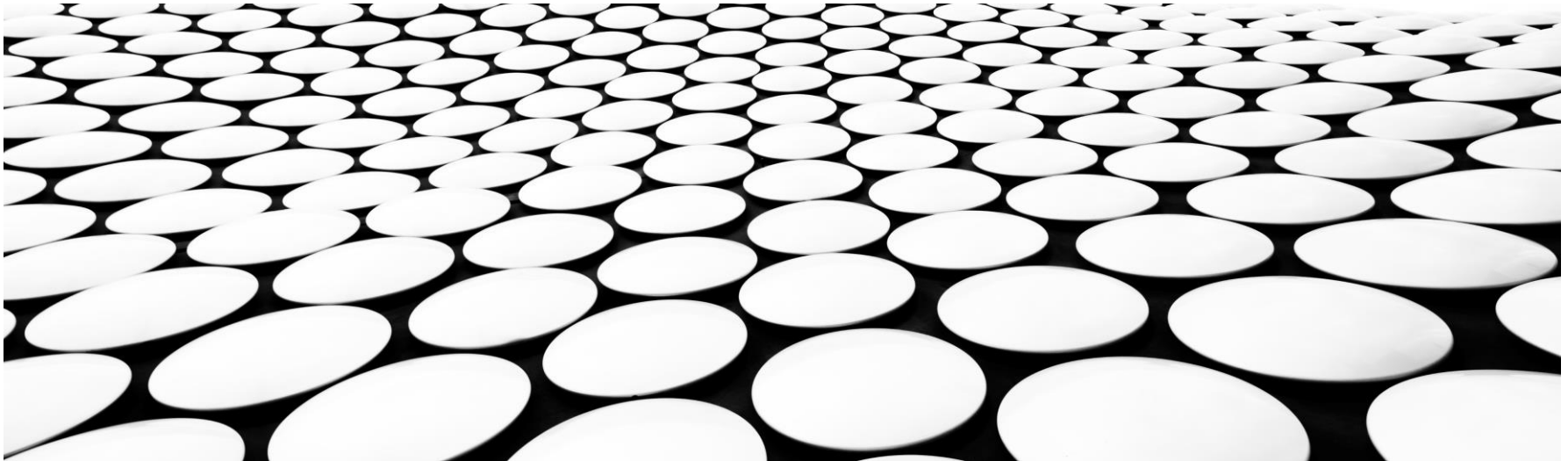
2022



كلية معتمدة من الهيئة القومية
لضمان الجودة والاعتماد

LECTURE 6

THE HELPER CLASSES , THE MULTITHREADED SERVER



C# SOCKET HELPER CLASSES

- The .NET Framework supports the normal socket interface for advanced network programmers, but it also provides a simplified interface for easier network programming.
- The *simplified socket helper classes* help network programmers create socket programs with simpler statements and less coding—two important benefits for any programmer.
- Here are the three helper classes used for socket programming:
 - [TcpClient](#)
 - [TcpListener](#)
 - [UdpClient](#)

TCPCLIENT

- The methods of the TcpClient class are used to create client network programs that follow the connection-oriented network model.
- The TcpClient methods mirror those in normal socket programming, but many of the steps are compacted to simplify the programming task.
- `TcpClient newclient = new TcpClient();`
- `newclient.Connect("www.isp.net", 8000);`
- The TcpClient class will automatically attempt to resolve the hostname to the proper IP address. That's a lot of work already done for you!

GETTING THE STREAM

- The `GetStream()` method is used to create a `NetworkStream` object that allows you to send and receive bytes on the socket.

```
NetworkStream ns = newclient.GetStream();
byte[] outbytes = Encoding.ASCII.GetBytes("Testing");
ns.Write(outbytes, 0, outbytes.Length);
byte[] inbytes = new byte[1024];
ns.Read(inbytes, 0, inbytes.Length);
string instring = Encoding.ASCII.GetString(inbytes);
Console.WriteLine(instring);
ns.Close();
newclient.Close();
```

TCPLISTENER

- the TcpListener class simplifies server programs; their class constructors are similar as well. Here are the three constructor formats:
 - TcpListener(int port) binds to a specific local port number
 - TcpListener(IPEndPoint ie) binds to a specific local EndPoint
 - TcpListener(IPAddress addr, int port) binds to a specific local IPAddress and port number

```
TcpListener newserver = new TcpListener(9050);
newserver.Start();
TcpClient newclient = newserver.AcceptTcpClient();
NetworkStream ns = newclient.GetStream();
byte[] outbytes = Encoding.ASCII.GetBytes("Testing");
ns.Write(outbytes, 0, outbytes.Length);
byte[] inbytes = new byte[1024];
ns.Read(inbytes, 0, inbytes.Length);
string instring = Encoding.ASCII.GetString(inbytes);
Console.WriteLine(instring);
ns.Close();
newclient.Close();
newserver.Stop();
```

UDPCIENT

- For applications that require a connectionless socket, the UdpClient class provides a simple interface to UDP sockets.
- You may be wondering why there is not a listener version of the UDP helper class. The answer is simple: you don't need one.
- Remember, UDP is a connectionless protocol, so there is no such thing as a client or server; there are only UDP sockets either waiting for or sending data. You do not need to bind the UDP socket to a specific address and wait for incoming data.

MULTITHREADING

Main Program

```
create Socket  
bind Socket  
listen on Socket  
while  
{  
    accept connection  
    create Thread  
}
```

Client Thread

```
Send welcome banner  
while  
{  
    receive data  
    send data  
}  
close socket
```



MULTITHREAD SERVER, WHY?

- A multithreaded server is any server that has more than one thread.
- Because a transport requires its own thread, multithreaded servers also have multiple transports.
- The number of thread-transport pairs that a server contains defines the number of requests that the server can handle in parallel.
- The normal server situation is that a single server handles many clients.
- Each client have its own handling thread , socket, and streams.

Server – high level view

Create a socket

Bind the socket

Listen for connections

Accept new client connections

Read/write to client connections

Shutdown connection

CLIENT – HIGH LEVEL VIEW

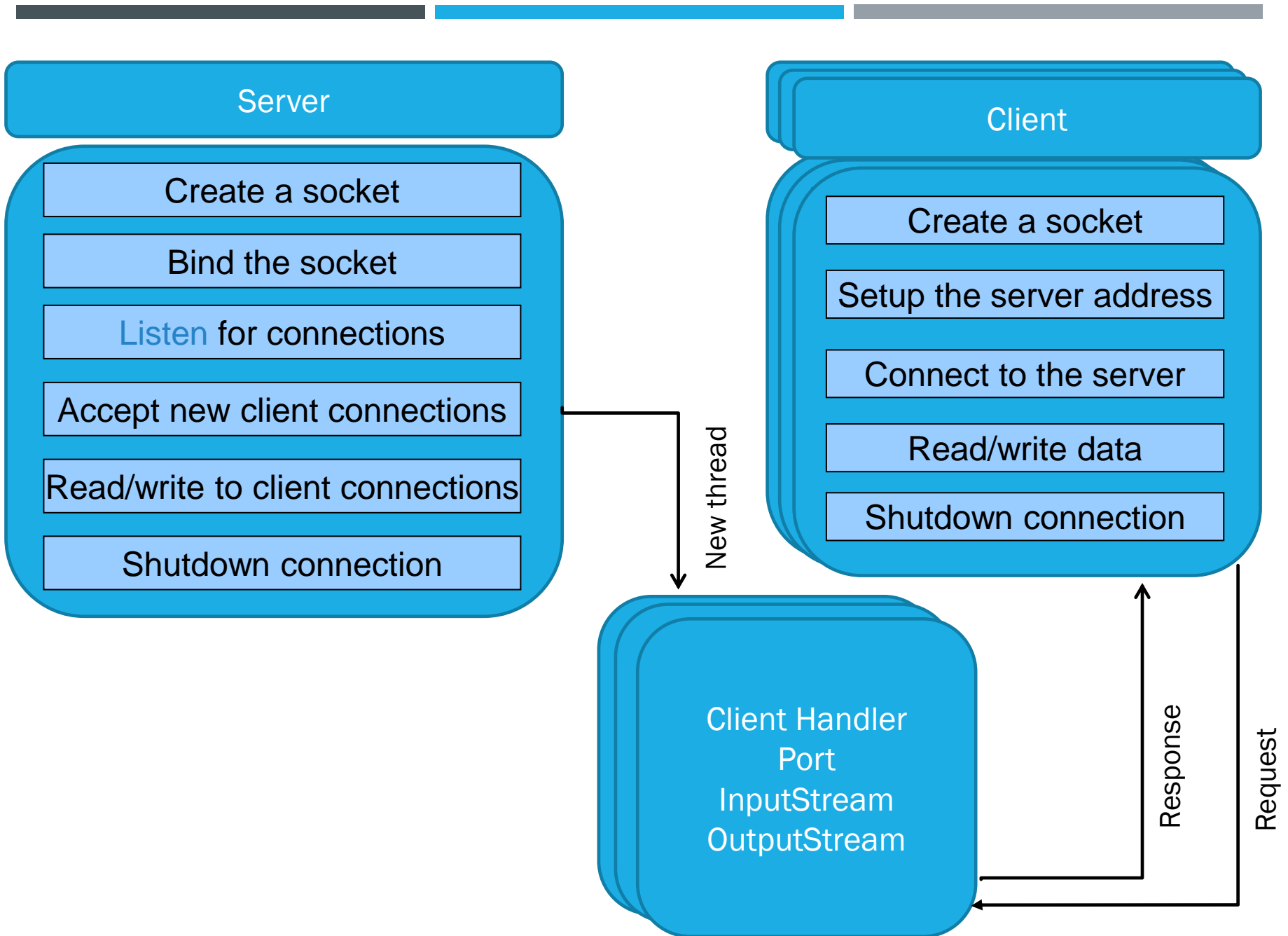
Create a socket

Setup the server address

Connect to the server

Read/write data

Shutdown connection



Time

To

code

THE MULTITHREADED SERVER

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
class ThreadedTcpSrvr
{
    private TcpListener client;
    public ThreadedTcpSrvr()
    {
        client = new TcpListener(8000);
        client.Start();
        Console.WriteLine("Waiting for clients...");
        while (true)
        {
            while (!client.Pending())
            {
                Thread.Sleep(1000);
            }
        }
    }
}
```

```
        ConnectionThread newconnection = new ConnectionThread();
            newconnection.threadListener = this.client;
            Thread newthread = new Thread(new
ThreadStart(newconnection.HandleConnection));
                newthread.Start();
        }
    }
    public static void Main()
    {
        ThreadedTcpSrvr server = new ThreadedTcpSrvr();
    }
}
```



```
class ConnectionThread
```

```
{
```

```
    public TcpListener threadListener;
```

```
    private static int connections = 0;
```

```
    public void HandleConnection()
```

```
    {
```

```
        int recv;
```

```
        byte[] data = new byte[1024];
```

```
        TcpClient client = threadListener.AcceptTcpClient();
```

```
        NetworkStream ns = client.GetStream();
```

```
        connections++;
```

```
        Console.WriteLine("New client accepted: {0} active  
connections",
```

```
                           connections);
```

```
        string welcome = "Welcome to my test server";
```

```
class ConnectionThread
```

```
{
```

```
    public TcpListener threadListener;
```

```
    private static int connections = 0;
```

```
    public void HandleConnection()
```

```
    {
```

```
        int recv;
```

```
        byte[] data = new byte[1024];
```

```
        TcpClient client = threadListener.AcceptTcpClient();
```

```
        NetworkStream ns = client.GetStream();
```

```
        connections++;
```


```
        Console.WriteLine("New client accepted: {0} active  
connections",
```

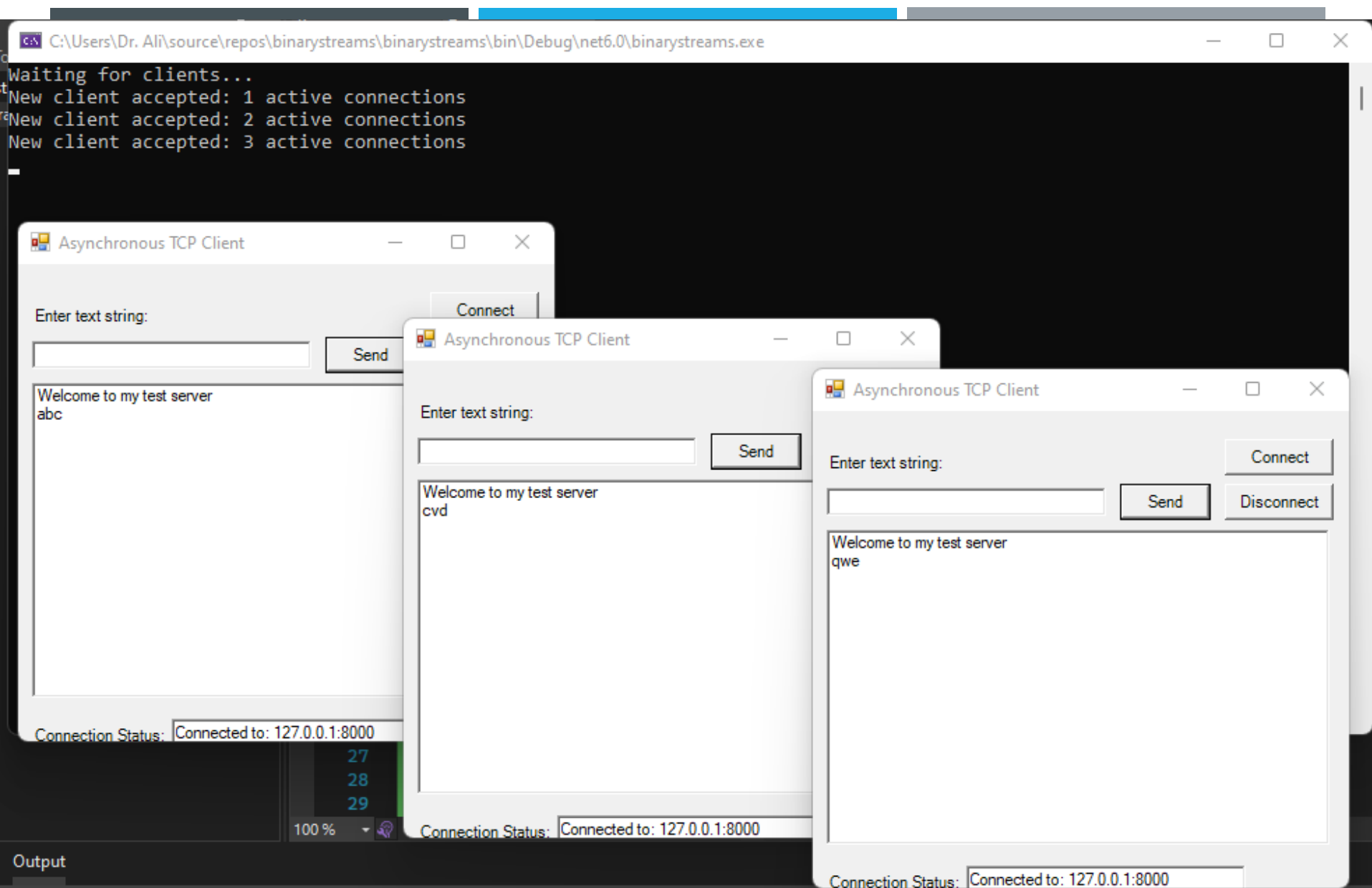
```
                           connections);
```

```
        string welcome = "Welcome to my test server";
```

```
data = Encoding.ASCII.GetBytes(welcome);
ns.Write(data, 0, data.Length);
while (true)
{
    data = new byte[1024];
    recv = ns.Read(data, 0, data.Length);
    if (recv == 0)
        break;

    ns.Write(data, 0, recv);
}
ns.Close();
client.Close();
Console.WriteLine("Client disconnected: {0} active
connections",
                connections);
}
}
```

- 
- The next step is to try any client (say the client from the previous lecture)





- Thank U